Toward Increased Retention in University Computer Science Programs
A Language Socialization Approach

By

CAITLIN MORIAH GREEN
B.A. (University of Washington)

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Linguistics

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA
DAVIS

Approved:

_____
Julia Menard-Warwick, Chair

_____
Nina Amenta

_____
Vaidehi Ramanathan

Committee in Charge

2018

ABSTRACT

This dissertation is a mixed-methods ethnographic study of students in the introductory class in the Engineering & Computer Science (ECS) department at University of California, Davis (UC Davis). I collected audio-recorded natural speech and questionnaire data from students attending this course during two academic quarters, Fall 2016 and Winter 2017. Between these two quarters, the instructor of the course introduced a new mandatory class activity that required students to speak to each other in order to complete it. Using this difference, I investigated the nature of peer socialization in Computer Science pedagogy by comparing the speech of students as they spoke to instructors to the speech of students as they spoke to peers. Using Language Socialization Theory and its related analytical methods, and employing Politeness Theory as a way to understand how students' understanding of academic and cultural concepts is mediated by conversational practice, I pursue workable recommendations to share with educators in CS and other engineering fields.

Instructors employed repetition as well as linguistic and metalinguistic coercion in order to provide sufficient stimuli to the learners as they worked to acquire new terms. Students speaking to teachers tended to avoid the use of class terms even when such avoidance, a violation of the Gricean maxim of quantity, created disfluency or confusion. These behaviors were supported and allowed by the instructors but not by peers. Non-expert use of terms was allowed by TAs and peers but not by the professor. Analysis of politeness demonstrated that students were much more reticent to impose on the instructors by asking for help, and that instructors' concern for the face needs of their students tended to impede their ability to facilitate learning in one-on-one interaction. Between students and instructors, I found frequent repairs, pauses, filler markers, mitigation and minimization, and indirect speech acts. Between peers, I

found frequent laughter, higher frequency of conversational turn-taking, boosting, and bald on-record strategies. Students speaking to peers used follow-up questions to ensure clear communications, which they avoided with instructors. When instructors asked students to volunteer to answer questions in front of the class, women tended not to answer at all. Students who did choose to answer did so in extremely short sentence fragments, and they often made use of high-rising terminal or question words. Students answering questions posed by peers spoke in complete sentences, using both on-record politeness and bald on-record strategies to do so, even when calling attention to a knowledge gap in their peers. Instructors, upon hearing student responses, would repeat their answers exactly, then provide commentary as to their stance on the answer. Students hearing a peer's answer would instead reword the answer to test his or her own understanding, seeking validation from their peer. Talk between peers seems to be accompanied by significantly less positive and negative face threat than that between students and instructors across several contexts within the ECS 10 course, and pair programming appears to require more active construction of knowledge from students than do activities involving speaking to an instructor.

Some discourses were recorded describing programmers as unkind and unlikeable. One core value of programmers, efficiency, was communicated via several diverse methods, but it appears to have been taken up by several of the students. It was also not typically accompanied by these discourses about what a programmer's personality was like, which may have contributed to students' relative ease of uptake. Some students, particularly women, who encountered difficulties, tended to describe those difficulties as a sign of their natural poor fit for the field. Increasing the amount of pair work students had to do may have decreased the effect of that tendency, as gender had less to do with grades and retention in the treatment quarter than in

the control. Asking instructors to make slight changes to their linguistic and discursive practices, alongside adding collaborative learning opportunities, is likely to create positive change in the learning of all students, particularly those in underrepresented groups.

ACKNOWLEDGMENTS

This dissertation is the result of years of work, and it was made possible by the support and advice of many brilliant and loving people. First, my major advisor Julia Menard-Warwick. I knew from the first course I took with her that she would be the person to guide me through this process. Julia, your insights and compassion have brought me through the highs and the lows of finishing this project. You mined your own professional life for the benefit of your advisees so that we could see academic research and writing in process, you created an atmosphere of trust and honesty for learners, and you moved heaven and earth to keep me on track, or, even when I wandered off track, to get me back. To Vaidehi Ramanathan, your brilliant comments on the prospectus and dissertation helped me understand my data better and represent it more clearly. Thank you for encouraging and challenging me. To Nina Amenta, you were on board immediately with my request to sit in on all of the activities of ECS 10. Your questions and ideas led me on the path to investigating the link between the linguistic, the cognitive, and the social. Thank you for helping me formulate my research questions and give me ideas for how to approach them, and for introducing me to Kurt Eiselt, the professor who taught the courses in this study. Kurt, thank you so much for creating a wonderful space for this research to take shape. You made every minute of it a joy, you facilitated the consent process and instructed the students as to how to participate, and you demonstrated so many excellent examples for me to derive my conclusions from. If other CS instructors emulate you in their practices, the field of CS education will be better for it.

To my parents and brother, Michael, Judith, and Sean Tierney, I owe everything. How many combined hours did we spend on the phone, talking while I drove between the Bay Area and Davis? How many times did I ask you to give me advice on the challenges I faced as a

graduate student and TA? More than I can count. Thank you for providing the love and support I needed during these five and a half years. To my husband, Evan Green: I am filled with joy and gratitude very time I contemplate your unwavering support and your ability to stay cool and calm even as I moved two hours away for two years, filled our tiny apartment with massive textbooks, and postponed our honeymoon to get myself through this. You encouraged me, you comforted me, and you listened as I worked through theories and challenges. You taught me to love cats, particularly one extremely squeaky black cat with an unnaturally long tail. You have filled our home with love, laughter, and more vintage technology than I will ever try to count, and I wouldn't trade it for anything. There is an alternate universe somewhere out there where you decided not to go to that mask party in 2010, and it's a universe I never want to see. You and your family have helped me in so many ways on this journey. I am so happy that you guys are my family, too. Go Team Green!

To my saltines, the twenty-one women who have made it their express mission to support one another through all manner of challenges: this dissertation is the direct result of your magic. The Amys (Alida and Hay), the Caitlins (Winks and Ayo Miss), Christine, Emma, Feffanie, Hannah, Jordana, Kimmy, KK, Laurie, Mel, Michelle, Monique, Ruth, Tams, Tracey, Turks, and Vanessa. Any time I faltered, any time I felt it would never get done, you were there. Each time I finished a chapter or an edit, you were there. You created a bottomless well of support, you taught me the slogan that kept me moving forward, and you reminded me that no matter what, I had a team behind me that I could never have imagined before I joined this group. Whenever I am lost, confused, or afraid, I just remember: Strong, Smart, Go!

To all of you and all of the wonderful friends, family, and colleagues who helped to educate, advise, support, and distract me as I worked on this project, a full-throated and warm thank you.

TABLE OF CONTENTS

ABSTRACT ii
ACKNOWLEDGMENTS v
TABLE OF CONTENTS viii
LIST OF TABLES x

CHAPTER 1: INTRODUCTION 1

OBJECTIVES AND RESEARCH QUESTIONS 2
Positionality 3
Computer Science in the Global Economy 5
Computer Science at UC Davis 9
Gender and Race Issues in Computer Science 10
Research Questions 17
CHAPTER OVERVIEW 19

CHAPTER 2: PREVIOUS SCHOLARSHIP 22

THEORETICAL FRAMEWORK: LANGUAGE SOCIALIZATION 22
Language Socialization Theory: A Summary 23
Identities, Stances, Ideologies 28
Language Socialization and Face Threat 37
Learning and Cognition: Legitimate Peripheral Participation 42
Collaboration and Pair Programming 47
Addressing Inclusion: Past Interventions 50
CONTRIBUTIONS OF THIS STUDY TO THE LITERATURE 54

CHAPTER 3: METHODOLOGY - DATA COLLECTION AND ANALYSIS 57

SETTING 57
The Lecture Hall 58
The Discussion Section 60
The Lab 61
PARTICIPANTS 63
Instructors 64
Students 67
RESEARCHER ROLE 68
DATA COLLECTION 70
Observation Types, Locations, and Times 70
DATA ANALYSIS 72
Ethnographic Analysis 72
Discourse Analysis 74
Conversation Analysis and Pragmatics 76
Statistical Analysis 81
CONCLUSION 82

CHAPTER 4: LANGUAGE SOCIALIZATION TO NEW TERMS 84
FUNCTIONS 85

Modeling "function" 85
Student uptake of "function" and related words 98
Pair Programming and Student Uptake 106
INDEX 111
Modeling "index" 112
Student uptake of "index" and related words 117
Pair Programming and Student Uptake 120
CONCLUSION 123

CHAPTER 5: POLITENESS THEORY AND NEGOTIATING MEANING 125

INSTRUCTORS WITH STUDENTS 125
Asking for Clarification: Positive and Negative Face Threat 126
Answering Questions: Positive Face Threat 132
PAIR PROGRAMMING ACTIVITIES 137
Asking for Clarification: Negative Face Threat 138
Answering Questions: Positive Face Threat 142
CONCLUSION 146

CHAPTER 6: SOCIALIZING PROGRAMMERS 149

WHO ARE PROGRAMMERS? 149
Efficient 151
Insular Nerds 158
Nerdy Nerds 161
STUDENT POSITIONING 164
Efficient 164
A Poor Fit for Programming 165
Unhappy/Tired 167
Appreciative of Good Code 170
GENDER, RACE, AND STUDENT SUCCESS 171
CONCLUSION 174

CHAPTER 7: CONCLUSION 176

RESEARCH QUESTIONS 176
CONTRIBUTION TO THE LITERATURE 184
IMPLICATIONS 185
LIMITATIONS AND FUTURE DIRECTIONS 188
CONCLUSION 190

APPENDIX A: TRANSCRIPTION CONVENTIONS 191
APPENDIX B: QUESTIONNAIRES 192
REFERENCES 196

LIST OF TABLES

Table 1: ECS 10 Socialization Events Summary ……. 57

Table 2: ECS 10 Discussion Sections Schedule …….. 60

Table 3: ECS 10 Lab Schedule …….....…… 62

Table 4: Demographic Information: UC Davis ……... 67

Table 5: Final score by self-reported gender and quarter enrolled …...……..… 172

Table 6: Choice to take more CS courses by gender and quarter ……..…….…… 173

CHAPTER 1: INTRODUCTION

This dissertation is a mixed-methods ethnographic study of students in the introductory class in the Engineering & Computer Science (ECS) department at University of California, Davis (UC Davis). This course, ECS 10, is titled Introduction to Programming, and the students learn to use a programming language called Python. I collected audio-recorded natural speech and questionnaire data from students attending this course during two academic quarters, Fall 2016 and Winter 2017. Between these two quarters, the instructor of the course introduced a new mandatory class activity that required students to speak to each other in order to complete it. This change allowed me to investigate the nature of peer socialization in Computer Science pedagogy by comparing the speech of students in the two quarters. Identity formation and learning new terms are both aspects of language socialization.

As students learn to program, they negotiate meanings in the process of incorporating new terms into their lexicons; some are completely novel terms and some are known words to which they must add new meanings. This dissertation takes several approaches to measuring and reporting on student learning: as a socially mediated process, which is measured by analyzing discourses in the classroom; as a discrete event accomplished through engaging with course material, which is measured by assessing individual utterances by students and instructors; and as an accomplishment achieved by the end of an assessment period, which is measured by course grades. Each of these perspectives allows for a different dimension as I describe the situation in which students find themselves as they learn to program.

The process of learning also includes identity formation, which has been described as it occurs in multiple fields of study but not in Computer Science (Rodriguez & Lehman, 2017). The identities present in the classroom can affect the ways in which learning takes place as well

as the students' ideas about their own fitness for the field of study. These identities include those performed by instructors and peers, as well as larger discourses that define the parameters of who is a natural fit for programming. In this chapter, I describe the objectives and research questions of the study on which this dissertation is based, and a chapter overview is provided.

Objectives and Research Questions

As the industry of software development continues to grow as an economic and cultural force, and as programming becomes an important feature of careers both inside and outside of that industry, the question of how best to prepare novices to the practice of programming has captured the interest of educators and researchers in diverse fields of inquiry (Shaashani, 1994; Mannix & Neale, 2005; Cheryan et al., 2009, e.g.). The introduction of computing concepts in early education contexts has been explored and tested, but of additional interest is the nature and methods of preparing students in higher education for careers involving programming. Educators and companies alike are interested in ensuring that a large number of qualified individuals enter the workforce in the future, and a key challenge to that goal is the underrepresentation of certain social groups. This is of interest to them for several reasons: first, more diverse teams work more efficiently and creatively than homogeneous ones (Mannix & Neale, 2005). Second, fewer competitors in a field result in less competitive applicants. For that reason, inclusiveness and best practices for maximal student preparation are both of interest for those concerned with improving the field of new software professionals. Because positive experiences in a student's first programming course drastically increase their intention to persist in the program (Shaashani, 1994; Beyer, 2014), this time frame is of particular interest. This research addresses gaps in existing knowledge concerning CS education by applying detailed ethnographic observation to

classroom activities in order to describe the process by which novices move toward full membership as programmers.

Positionality

As an undergraduate university student in Computer Science courses in the early 2010s, I found that I was one of the only female students in most of my courses. The other women I met became highly valuable to me as sources of emotional and academic support, and I still keep in touch with them. I was not a student in the Computer Science department but some of my classes required me to spend time in their lab. I was highly uncomfortable entering that space without a female CS student with me because I sensed that I did not belong. Cheryan et al. (2009; 2011) suggest that what I was sensing was a lack of *ambient belonging*: the space communicated to me that I had very little in common with those who more typically occupied it. After our undergraduate education, those of my friends who had completed Bachelor's Degrees in CS had very little trouble securing positions at tech companies. Many of the women, however, did not stay there. Citing various reasons including loss of interest in the field, dissatisfaction with the work environment, and a sense of being creatively restrained by large tech companies, many of the women I knew moved into other roles or other careers entirely.

As I began to pursue research in the area of language and gender, I discovered that discourses of "self-selecting" were being used to describe the phenomenon of women leaving STEM (science, technology, engineering, and mathematics) programs and positions. Many of the programs I encountered whose stated purpose was to increase female participation in STEM pointed to women's lack of understanding that STEM really was for them. I began to wonder whether this truly was a lack of understanding on women's part or shrewd detection that they

were entering a domain in which they would be consistently reminded that they were a minority. It influenced me to look into discourses around gender and technology, considering these discourses to be evidence of a problem whose locus was not solely in the minds of the women who were leaving. In later courses, I looked around myself and noticed that not only were female students underrepresented, but that the only students of color in the room were largely Asian and Asian-American students, who experience a "model minority" stereotype in the United States (Kao, 1995) and are therefore much more numerous in programs connected to high academic achievement and lucrative careers. Black and Hispanic students were much less numerous than white female students like me. I wondered how these and other marginalized students could be made to feel more welcome and more successful in the program.

Many of the reasons given for this attrition was expressed in terms of community and belonging, and many commonly held beliefs were expressed through similar discourses. Because my research interests include language socialization, discourse analysis and gender theory, I believed that I could help search for the answer to this question. During my graduate studies at UC Davis, I enrolled in an CS course despite my discomfort at feeling different and apart from my classmates, and I enrolled in a series of online Python courses with Rice University to become fluent in the concepts and terminology of that programming language so that I could observe and assess the usage of Python-related terminology in my own research. I reached out to the Computer Science department at UC Davis and discovered that they had some of the same questions as I had. I worked with CS department chair Nina Amenta to develop some preliminary research questions on which this study is built.

Computer Science in the Global Economy

The goal of an introduction to programming is to teach students how to design and write programs. Typically, each course of this type focuses on one programming language to do so. A program is "a sequence of instructions that specifies how to perform a computation" (Downey, 2015; p. 1). The definition of a "computation" in this case is somewhat flexible; programs can be used to provide the answers to mathematical questions, but they can also take in documents, images, and sounds, and they can formulate conclusions or execute actions based on these inputs. The physical act of programming entails typing out instructions in a style and order that the computer will understand and execute. Just as a writer has their choice of word processor, where one types these instructions is a matter of preference for the programmer; what computer application reads these instructions depends on the language used and the desired effect. Some prefer to use simple text editor applications, such as TextEdit, TextWrangler, or Notepad. Some text editor applications only allow for text to be entered; they do not interpret that text as code. Some applications, such as TextWrangler or SublimeText, can be instructed that a particular programming language is being used and applies colors to important elements of that code. This will be discussed in more detail in Chapter 2. Once the program is typed into these documents, the files are saved with a suffix corresponding to the programming language in which it was written. In the case of a learner of Python, that suffix is .py. This means that a program called "my_program" will be saved as "my_program.py", much the same way that a Word document called "essay" would be saved as "essay.docx". Many programmers choose to use what is called an interactive development environment (IDE). These are applications that contain a text editor and an interpreter, a machine that interprets and executes the program to demonstrate its function. The interpreter is sometimes called a shell, and all computers have a shell that can be

used with or without a development environment. The participants in this study used an interactive development environment called IDLE, which was designed specifically to function with Python. In order to create a new program, programmers must open a new file, write in it, and save it with an appropriate name. That file can be accessed by IDLE and the program can be run, which means that the interpreter follows all the instructions contained in the program. The results of the program can be seen in the interpreter, which allows the programmer to make necessary changes.

The participants in this study learned to use Python, which is a programming language invented in the late 1980s by Guido van Rossum, who worked at Centrum voor Wiskunde en Informatica (Klein, 2011). Programming languages are more rigid in their structure and less complex than natural languages, but they have lexicons and syntax that dictate how their elements can be combined to create programs that can accomplish nearly anything a programmer can imagine. Python in particular is a language that has a relatively basic syntax, few punctuation marks and even fewer data types, leading people to refer to it as an easy language to read and a logical first language to learn. Guido van Rossum named the language Python after the UK comedy group Monty Python, as a nod to the revolutionary effect he hoped this simpler, more readable language would have (Klein, 2011). Readability, simplicity, and beauty are explicitly stated values for van Rossum and those who teach his language. Over the course of the 1990s, the language was published, adjusted, and re-published. It became popular in the early 2000s with the release of Python 2.0, and now programmers are moving to Python 3.0. Python is the fastest-growing programming language in the world: on Stack Overflow, a popular programming advice forum, inquiries about Python make up about 64% of the traffic originating in high-income countries such as the United States, United Kingdom, Germany, and Canada (Robinson,

2017). The trajectory of Python's rise in developer posts far outpaces those of Javascript, Java, C#, PHP, and C++. As of June 2017, Python was the most visited tag on Stack Overflow within high-income nations, possibly due to its flexibility and readability.

Those who graduate with a degree in CS or demonstrate sufficient experience to find employment in the CS field tend to become software developers or project managers, although a CS degree is not necessarily required for these positions (USNews). Programming requires fluency in several programming languages. More job openings are expected to materialize than there are qualified programmers (Lockard & Wolf, 2012), so while finding a job with a prestigious company is challenging, finding programming employment in general is relatively secure. While some may define "programmer" and "software engineer" as equivalent, others consider software engineers to be more involved in big-picture program architecture. An entry level software engineer can expect to earn an average salary of $76,215 in California (Indeed.com), with stock options and bonuses on top of that amount, and they can expect frequent and sizeable raises as they persist in a company. Software development is required by a wide range of industries, most notably technology companies but also in finance and government (Lee, 2017). Smartphones and other mobile devices are extremely prevalent, and programmers are needed to design, implement, and update the applications that run on them. It is expected that mobile app development will be a significant proportion of programming careers for some time. Web development has constituted a large portion of programming careers since the advent of the Internet and will likely be required quite some time: web development consists of front-end work, such as design and user interactive features, and back-end work, which manages the relationship between the data in the site and the Internet in general. Web development and other

careers may involve knowledge of databases, which requires querying large caches of data of many types and using or manipulating that data as needed.

In California, many programmers live in northern California, especially in Silicon Valley, the area immediately surrounding San Francisco. Mountain View is home to Google, Apple's headquarters are in Cupertino, Intel is in San Jose, Menlo Park houses Facebook, and many other tech giants and startups call this area their home. This area is frequently criticized as a site of extreme economic inequality, exclusionary displacement, and unchecked consumption (Robertson, 2017; Kendall, 2018; Taylor, 2018). Salaries are high because of a housing shortage and because the cost of living in this area has risen much more quickly than that in other parts of the United States (Avalos, 2018). Because of their significant presence in the area, programmer culture is a known reality among northern California residents, regardless of their industry of employment. The precise discourses around programmer culture will be described in detail later in this chapter, but one can see by visiting the campus of a tech company that its cultural values include celebrating nerdiness and encouraging employees to spend as much time as possible there. The idea of the all-encompassing workplace has begun to enter the popular culture in television shows like Silicon Valley (HBO) and the film *the Circle* (2017) based on the novel of the same name (Eggers, 2015). Characters walk the halls and walkways of massive compounds, passing gyms, cafeterias, meditation areas, and even dance clubs, filling viewers with the creeping feeling that when companies fill its employees' every need, it creates a sense of dependency and even isolation from the rest of the world. This sense creates a kind of harmony with the cultural concepts of programmers, which are very connected to nerd culture, a cultural concept that involves obsessive interest in technology and very little engagement in social activities (Gershuny, 2003; Tobin, 2011). The combination of the economic power, the sheer

number, and the nerdiness that are indexed by the programmer identity results in a complex and challenging phenomenon that will be discussed in detail as it relates to goals of inclusion and socialization.

Computer Science at UC Davis

UC Davis is a large public university in northern California. Part of the University of California system, UC Davis was attended in 2016 by approximately 29,000 undergraduates, 6,700 graduate students, 11,800 faculty members, and 23,000 staff members (UC Davis Student Profile; UC Davis Employee Summary Data). The department of Engineering and Computer Science (ECS) resides within the College of Engineering and houses approximately 1200 undergraduates, 200 graduate students, and 41 faculty members (CS People). To earn a major in this department, students must complete courses in mathematics, natural sciences, and engineering, followed by courses that deal with computer software and hardware (Computer Science and Engineering). ECS 10, Introduction to Programming, is not required by the major, but many students choose to take it before attempting ECS 30, which has a reputation as an extremely challenging class (Salanga, 2018). Students hear that ECS 10 is easier to pass and will help them decide whether programming is right for them. Many other science and mathematics-related majors require some ECS courses.

In my meetings with Professor Nina Amenta, we discussed concerns over terminology acquisition in the first quarter of the program. Because UC Davis uses the quarter system, each course only lasts ten weeks. In this time, students in ECS 10 must learn a large variety of new terminology and concepts at the same time that they are learning what a program is and how to write one. Together, Nina and I created some preliminary research questions that would allow

me to use linguistic methods to assess strategies that worked and devise useful recommendations for teachers and administrators in the CS field, or those in other Science, Technology, Engineering and Mathematics (STEM) fields. Once I was committed to studying the introductory course, Professor Amenta brought in Kurt Eiselt, the professor who would be teaching it. He and I refined the research questions further and designed an experimental intervention in which the second group of students would be asked to work in pairs during some of their discussion sections. While pair programming was initially designed as an onboarding practice for new employees, i.e., people who already knew how to program but needed to learn to work within a company-specific system, Kurt suggested, and I confirmed in my review of the academic literature, that it has been found to be effective even for very early beginners. I also raised an issue of concern for me, the shortage of women and people of color (particularly Black and Latinx groups) in the CS industry, an issue that will be discussed in detail in the following sections. Professor Amenta and Kurt both expressed interest in findings pertaining to identity formation and inclusiveness as well, so this was also added to the research questions I formulated going forward.

Gender and Race Issues in Computer Science

The issue of lack of diversity in the field of software development has been a central area of debate in the California Bay Area particularly and in the United States in general; more people each year earn a Bachelor's degree in computer science, but only 22.3% of 2011 Computer Science (CS) Bachelor's degrees were conferred upon women (NSF, 2013), and that number is even declining (Katz et al., 2006; Miliszewska et al., 2006; NSF, 2017). In a survey of first-year college students in 2011, only 0.4% of the women intended to major in CS, compared to 2.9% of

the men, more than seven times as many (NSF, 2013). Attrition for women in CS programs is 32%, which is also higher than that of the men who pursue the same course of study (Cohoon & Lord, 2006). The women who do complete a CS degree report feeling less certain they will pursue a career in programming (Unfried et al., 2014; Lehman et al., 2016) even though CS careers are extremely lucrative compared to those in other fields (Lockard & Wolf, 2012). More members of underrepresented ethnic minorities have been enrolling in computer science undergraduate programs each year (NSF, 2013) but the gap between these students and white students continues to grow. Racial and ethnic minority students continue to earn fewer bachelors and doctoral computing degrees than White and Asian students (NSF 2017). At UC Davis, undergraduate enrollment in either Computer Science or Computer Science & Engineering rose from 492 to 615 between 1995 and 2004, an increase of 25% (First Majors by Discipline) . Undergraduate enrollment in the College of Engineering rose from 2,612 to 4,255 between 2006 and 2015, an increase of nearly 63% (Students Assessed Undergraduate Level Fees). Demographic information as a function of major was not available from the institution.

In introductory CS courses, gender is a significant factor in students' decision to persist in the CS field; men are more likely to take another course after the first one than are women (Barker, McDowell, & Kalahar, 2009). Students report that the first year, even the first semester, is make-or-break in terms of whether they will be able to finish the program (Shashaani, 1994; Beyer, 2014). In fact, female students and students of color are only as likely as white male students to drop out of the program once enrolled in the major (Lord et al., 2009). For women studying CS, a positive experience in the first course can make them more likely to persist in the major (Beyer, 2014). This means that if universities hope to make a change in their ability to recruit female students, students of color, or students of another marginalized group into the

Computer Science major, they must make this change before the end of the first Computer Science course.

More diversity in CS workplaces is important to the field for several reasons. It is predicted that there will be more CS jobs than there are qualified applicants in the near future (Lockard & Wolf, 2012). Diversity increases creativity and innovation on development teams (Mannix & Neale, 2005). Companies that have a more diverse staff in both upper and lower levels tend to have increased sales revenue and profits (Herring, 2009). The underlying reasons for low participation and the possible solutions are hotly debated in popular media, government, and academia. Researchers from diverse academic traditions have contributed to this debate using a wide variety of methods and theoretical backgrounds including cognitive psychology, social psychology, feminist theory, and pedagogy, among others. Women and students of color do not have worse grades or ACT math scores (Tam & Basett, 2006), suggesting that the issue does not arise from different levels of academic achievement. The software industry is associated with high salaries (Lockard & Wolf, 2012) and has a smaller gender pay gap than other industries (Weinberger, 2006)., suggesting that lack of financial opportunity is not likely the cause either. Research considering cultural and psychological measures suggests that the low enrollment and high attrition of women and students of color more likely comes from without than within.

Questionnaires and discourse analysis regarding cultural identities suggest that part of the problem has to do with nerd culture and programming culture (Fisher, Margolis, & Miller, 1997; Hapnes & Rasmussen, 1998; Kendall, 2000; Tobin, 2011, e.g.). The term "nerd" originally appeared in the 1960s, meaning a solitary person who works with numbers and is obsessed with technology (Gershuny, 2003). Asked to define what a nerd is, participants described a "young

man who is pasty, badly dressed, poorly groomed, has few friends, and loves Star Trek, violent video games, and, most of all, computers" (Tobin, 2011, p. 504). They have no time for parties (Bracey, 1993), instead preferring to spend all their time on the single-minded pursuit of programming (Hapnes & Rasmussen, 1998; Margolis & Fisher, 2002). While women can certainly be nerds, nerd culture is distinctly white and masculine (Bucholtz, 1989; Fisher, Margolis, & Miller, 1997; Kendall, 2000; Clegg, 2001; Oldenziel, 2001; Margolis & Fisher, 2002; Flowers, 2018). Based on questionnaires, interviews, and research of American popular culture, researchers have argued that "nerd" is a primarily masculine identity that is developed as a response to hegemonic masculinities predicated on physical prowess or social power (Connell, 1990). This technical masculinity also creates an expectation that while social graces are not required or even expected of nerdy men, the importance of the functions they perform make them indispensable (Kendall, 2000). Masculine identities that do not include these elements are considered less sexually attractive (Kendall, 2000), driving the formation of an identity that is highly protective of the technical competency it does contain and even imbuing that technical competency with straight male sexuality (Kendall, 2000). Women, who by definition do not identify with straight male sexuality, are often considered unratified and even not "real" members, and their contributions to the field also considered less important (Kendall, 2000). Those who perform masculine identities are rejected socially, and those who perform feminine identities are rejected professionally. In television and film, making a nerdy character hyper-feminine is a shortcut to humor, because it is considered a bizarre juxtaposition (Kendall, 2000). Nerd culture is also accused of normalizing sexual harassment and assault, as nerdy characters are portrayed as harmless or even admirable for committing such crimes (Kendall, 2000; Sils et al., 2015; Salter & Blodgett, 2017). This identity and its representation in popular media has

become more important to those hoping to increase diversity in CS programs and careers because of the significant increase in mainstream popularity that nerd culture has enjoyed (Stanley, 2015; Salter & Blodgett, 2017).

Programmer culture itself also comes with several problematic features. The most consistently referenced aspect of programmer culture in studies across disciplines is that true members involve computers and programming in all aspects of their lives (Hapnes & Rasmussen, 1998; Margolis & Fisher, 2002). The idea of addiction is used with a lighthearted mood (Hapnes & Rasmussen, 1998; Kendall, 2000; Margolis & Fisher, 2002; Beyer, 2014) and this all-consuming fascination is believed to last from early childhood through adulthood (Margolis & Fisher, 2002). Men who work in programming report spending most of their time thinking about code since they were children and consider the computer lab at school to be the most accepting and fun place to be in their spare time (Hapnes & Rasmussen, 1998; Margolis & Fisher, 2002). Workplaces in Silicon Valley provide all of life's needs: beds, workout equipment, food, and healthcare can all be accessed at work, and there are enough social events on campus to keep employees from ever needing to leave (Margolis & Fisher, 2002). Women also report perceptions that Computer Science entails too sedentary and isolated a lifestyle (Carter, 2006; Khoja et al., 2012).

Programmer culture is also associated with a sense of joy at working on computers (Hapnes & Rasmussen, 1998; Kendall, 2000; Margolis & Fisher, 2002; Leder & Vale, 2004). Survey-based studies have resulted in findings that girls and women tend to have lower interest in computers as objects than boys and men do (Shashaani, 1994; Shashaani, 1997; Carter, 2006; Robertson, 2013). Boys are more likely than girls to express pleasure about the prospect of doing an activity on the computer (Leder & Vale, 2004), which is unsurprising given that respondents

to questionnaires consider it "natural" to have boys play on computers and girls play with toys related to cooking, fashion, or social events (Margolis & Fisher, 2002). This expectation reinforces itself because parents buy toys for their children that reflect their own beliefs about what those children will be best suited to doing (Margolis & Fisher, 2002). Teenaged and adult girls, when asked, report that rather than seeing programming as a hobby, it is a way to find lucrative work or an important tool for excelling in a different field such as design or data analysis for other sciences (Margolis & Fisher, 2000). Girls who took an early interest to computers report the sense that other people considered them odd (Margolis & Fisher, 2002). In the computer lab and in CS classes at middle and high school, girls received unsettling comments about their bodies and appearance, were asked if they had come to the wrong room, and heard jokes suggesting they would be better suited to sewing or other stereotypically feminine pastimes (Margolis & Fisher, 2002). There are other cues that computer culture is reflective of masculine values (Clegg, 2001; Oldenziel, 2001; Sele, 2012). A popular piece of children's literature about what adults do claimed that men design computers and women use them (Oldenziel, 2001). At a large public university in the 2000s, there were two introductory-level courses being run at the same time: one was an information technology class and the other discussed social aspects of technology (Allen & Thomas, 2006). The information technology class was composed of 20% women, while the social issues class contained 70% women. Despite the existence of many crucial women in the history of programming, when asked to name a female role model 56% of students in an information technology class could not name a single one, even a fictional one (Allen & Thomas, 2006). 60% of these students did not know of any women working in CS. In a questionnaire, women expressed the perception that most computer scientists are men (Khoja et al., 2012). Introducing more women role models is discussed as a positive step (Dennehy &

Dasgupta, 2017), but some research suggests that doing so loses its effectiveness if the woman in question is perceived as nerdy (Cheryan et al, 2011).

Another hypothesis as to the reason women may be less well-represented in the field of CS is raised by psychology. Women also express less confidence than men even when conditions are controlled for prior experience in Computer Science (Shashaani, 1997; Beyer et al., 2003; Doubé & Lang, 2011). On an index of fear of failure, Newman, et al. (2013) found that female students scored much higher for Fears of Experiencing Shame and Embarrassment, Fears of Devaluing One's Self-Estimation, and Fears of Having an Uncertain Future. Women also showed lower self-efficacy (Doubé & Lang, 2011), which can harm academic outcomes, cause a negative feedback loop, and drive students toward failure (Linshinsky et al., 2016).

Beyer et al. (2003) relate this high occurrence of fear to *stereotype threat*. Many researchers in the field of social psychology point to stereotype threat as a possible cause for a range of issues including lack of interest, low performance, and lack of perseverance in stigmatized groups (Inzlicht & Ben-Zeev, 2000). Stereotype threat is being at risk of confirming a negative stereotype about a social group through one's own actions (Steele & Aronson, 1995). It has been shown to affect women in math classes (Good et al., 2008). Stereotype threat can affect performance in CS courses (Spencer et al., 1999; Huff, 2002) as well as course selection (Eccles et al., 1999). In fact, students from underrepresented groups report perceiving slightly more racism and sexism in CS classes than well-represented students (Barker, et al., 2009). Steele & Aronson have documented this effect in people of color but other studies suggest that women interact with stereotypes in a similar way. Stereotype threat can cause significant stress, and rumination about these risks takes up valuable working memory and impairs cognition and decision-making. While female students do not tend to perform poorly in CS programs overall,

the mere presence of men (Inslicht & Ben-Zeev, 2000) or the reminder that gender differences exist in performance of a task (Spencer, Steele, & Quinn, 1999) can impair women's problem-solving abilities. Women under stereotype threat are more likely to over-use previously successful but presently inadequate problem-solving methods due to inelastic thinking (Carr & Steele, 2009), which may be the result of *stereotype suppression*, or the hard work of actively stopping oneself from thinking about the stereotypes that are threatening one's success. This danger may also be a cause of female attrition: for example, women who earn less than a B in an introductory programming course are less likely to continue to the next course than men who earned the same grade (Katz et al., 2006). This could mean that women take mid to low performance as confirmation that they do not belong in the program more often than men do.

A related concept in psychology is called *ambient belonging* (Walton & Cohen, 2007; Cheryan et al., 2009; Cheryan, Meltzoff, & Kimm, 2011; Cheryan et al., 2011; Cundiff, 2013). If a person enters a space in which they believe they will find few friends, their achievement and interest in participation will decrease. This belief can arise through discourses that construct the typical computer scientist as a white male nerd, stereotypically nerdy or masculine physical objects in a space, or a similar design for a virtual classroom environment. The measures that have been taken to counteract challenges to women and people of color in CS courses, and their relative effectiveness, are discussed in Chapter 2.

Research Questions

My research interests focus on how identities are performed as individual events of *positioning* in larger sociocultural contexts, which refers to the ways in which people make use of commonly understood social representations to implicate themselves and others as being

members of relevant social categories (Davies & Harré, 1990). Language socialization treats these events as steps in the process of inventing oneself as a full member of particular communities. In concert with this process of invention is the construction of linguistic systems that allow novices to begin to participate fully in the target community. In meetings with Kurt and Professor Amenta, I developed the following research questions in order to describe the phenomenon of identity formation and terminology learning:

1. In what ways do instructors socialize students to use programming-related terminology? In what ways do students take up those terms?
2. What is the effect of increasing student collaboration on classroom talk, particularly the ways in which students use terms and achieve important speech acts relevant to learning?
3. What discourses and identities do instructors construct as they socialize their class as members of a culture of programmers?
4. In what ways do those students support, reconstruct, or resist those discourses and identities?
5. What is the statistical relationship between quarter (control group versus treatment group), academic performance, and plans to continue in the program?

In the pursuit of answering these questions, I observed classroom activities, lectures, and individual laboratory tutoring sessions. I also collected questionnaire data from students about their attitudes toward the course and instructors as well as data describing their academic performance and applied descriptive statistical methods to these values as a function of race and

gender. The results of this data is described within this dissertation and situated within a larger cultural context of programming education and professions.

Chapter Overview

This chapter has described the aims of this dissertation in terms of the linguistic practices that characterize early student engagement with programming. In this chapter I summarized the cultural context in which students considering a career in CS find themselves. I described the issue of inclusion and diversity, its cultural salience, and some theories that have been applied to attempt to improve the situation. The research questions that guide this dissertation are listed and discussed.

In Chapter 2, the previous scholarship relating to language socialization is reviewed. This chapter takes the linguistic practices of instructors and students and couches them in Language Socialization Theory, which is used to situate ideas concerning pedagogy, identity formation, and collaboration. I discuss the applications of Language Socialization to learning, cognition, identities, and larger cultural discourses, drawing connections between individual acts of socialization and the larger ideologies they reflect and support. In order to situate Language Socialization Theory in the Computer Science classroom, I describe research that has been undertaken to investigate the practices, identities, and ideologies involved in teaching people how to be programmers. An important analytical tool employed in this process is *face threat* and *face threat redressive strategies*. Building from the foundational work of Brown & Levinson (1978), I analyze the ways in which choices in wording, gesture, tone, and volume can mediate the effects of an utterance on a hearer, constituting management of *face,* or the perception of a hearer that they are well-regarded and free to take actions relatively unencumbered.

The methods by which data was collected and analyzed are described in Chapter 3. The multiple settings in which learning activities took place are described in detail, as are the participants in this research. I discuss my role as a researcher and the many identities I found myself performing over the course of the data collection period. I describe the theoretical framework governing my data collection, including ethnographic practices and questionnaire design. In this chapter, the practical aspects of entering the physical spaces and collecting data are discussed. The methods by which I analyzed the collected data is discussed in detail: First, ethnographic analysis, which is used to create a representation of the social system the participants constructed; second, discourse analysis, which provides a sense of the beliefs participants had concerning the nature of programming and what kinds of people could be programmers; third, pragmatic analysis, which allows a granular view of the many interactional negotiations that make up a socialization event; finally, statistical analysis, which is applied to questionnaire and registrar data to describe the measures of success set out by the experimental dimension of this study.

The data concerning how terms were modeled and taken up in class is analyzed in Chapter 4. The focal terms here are "function," which was introduced early in the ECS 10 course, and "index," which was introduced later. Language Socialization Theory is used to describe the instructors' methods for modeling the use of these terms, and I compare students' behavior around these terms between when they are in the presence of an instructor and when they are in the presence of a peer. Issues of power imbalances as a result of the expert-novice relationship are discussed as a potential explanation for the difference in behavior in these two contexts.

Chapter 5 discusses the relationship between power, facework, and students' help-seeking and informing acts. Taking inspiration from the findings in Chapter 4, this chapter takes a detailed look at the potential face threat involved in two focal speech acts: asking and answering questions. These behaviors are described first in student-instructor dyads, and then in student-student dyads, and comparisons made between them. The power differential posited as salient in Chapter 4 is described in terms of positive and negative face, analyzing the amount of facework students engaged in depending on their interlocutor as an indicator of power distance.

In Chapter 6, identity formation is discussed in detail. Discourse analysis techniques are used to demonstrate the transmission of culture between participants. The sociocultural concept of "programmer" is discussed in terms of the imagery and linguistic practices employed by instructors to construct an exemplar that is insular, nerdy, and efficient. The extent to which students reinforce or resist these discourses is discussed. The statistical measures of academic performance and persistence are discussed as a factor of gender and race, as well as the experimental intervention of pair programming.

This dissertation concludes with Chapter 7, which reviews the research questions established in this chapter, positing answers for each. It connects these findings to the existing scholarship relating to language socialization, identity formation, and inclusion in programming. A discussion of the implications of this study follows, focusing on generating a set of best practices that can be put into motion in CS programs at UC Davis and beyond, even generalizing to other areas of study. The final chapter describes the limitations of this study and makes suggestions for future research to help address some of those limitations.

CHAPTER 2: PREVIOUS SCHOLARSHIP

This chapter reviews scholarship relevant to the current study of Language Socialization in Computer Science education. In this chapter, I summarize the genesis and evolution of Language Socialization as a framework for analyzing how individual acts constitute the social construction of knowledge. This knowledge includes how to speak as a member of a community, and how learners construct, support, and resist the identities, ideologies, and discourses that a community considers culturally relevant. To relate the process of language socialization to other ways of understanding how cognition develops in learners, I also employ situated learning and legitimate peripheral participation, which describe how learners construct knowledge alone and with one another. I summarize relevant findings regarding how Language Socialization takes places in higher education, and specifically those examining learning in fields relating to Science, Technology, Engineering, and Mathematics (STEM), connecting it to research regarding pedagogy for Computer Science (CS) and its relationship to theories of learning, as well as its applications to the treatment that the students in Phase Two of this study underwent, which was the introduction of pair programming. Finally, this chapter closes with an argument that analyzing Computer Science learning from an interactional perspective will benefit the field of CS education by yielding actionable conclusions that instructors and administrators can use to improve student outcomes and experiences.

Theoretical Framework: Language Socialization

Language socialization is a theoretical framework that is particularly interested in the sociocultural knowledge involved in becoming a full member of a community. It is concerned with how experts negotiate their position as the guardians of proper behavior, how novices

reproduce, resist, and transform those behaviors, and how their active participation in a community constitutes their membership in it. As humans are social beings, engaging with the world largely means engaging with other people, which requires analytical tools that deal with the interpersonal face negotiations that take place in social contexts. Facework and Face Threat describes how people manage relationships with one another while also executing needed speech acts (Goffman, 1967). In the case of classroom conversation, this involves declaring the truth of propositions, asking questions, answering questions, giving advice, correcting, among others. In this section, Language Socialization, its application to cognition and learning, and its application to identities and discourses are discussed.

Language Socialization Theory: A Summary

Language Socialization Theory began as separate from language acquisition (Ochs & Schieffelin, 2017), which described children's language learning as a set of psychological tools by which children learned the elements of those languages in a set progression (Bloom, 1970; Brown et al., 1968, e.g.). Language socialization was concerned with the ways in which children practiced cultural and social skills needed for adulthood by engaging with their social surroundings (Whiting et al., 1975, e.g.). As ideas about learning and the mind changed, language socialization began to take as a unit of analysis the *speech community* (Gumperz, 1968) and measuring the degree to which members of that speech community demonstrated *communicative competence* (Hymes, 1972a), the ability for participants to engage with one another in socially appropriate ways. Discrete instances for analysis are called *speech events* (Hymes, 1972a, b; Duranti, 1985): these are activities that the speech community recognizes as a salient practice within their culture, and they are considered to have appropriate settings, participants, and acts and which require appropriate communicative abilities to execute.

Language socialization does not end when a person becomes fluent in their first language(s); each new domain and social group brings with it certain new language features and practices (Ochs, 1986; Ochs & Schieffelin, 2011; Duff, 2017). This means that language socialization continues through adulthood as we meet new people and join new communities, be it by moving house, attending a new school, joining a new field, or any other change in social landscape (Ochs, 1991; Duff, 2017). While much language socialization research is focused on young children learning to speak (Ochs & Schieffelin, 1986; Garrett & Baquedano-López, 2002; Blum-Kulka, 2017), it is often applied to adolescents (Brice Heath, 2017) and even adults (Hobbs, 2004; Duff, 2017) who are learning to use new languages (Jupp et al., 1982; Matsumura, 2001; Minegishi Cook & Burdelski, 2017) and to use their own language in new ways (Jacobs-Huey, 2003; Duff, 2017). Language Socialization research is turning its eye to those learning oral communication skills for practices such as presentations, mini-lectures, group project work, and class discussion (e.g., Duff, 1995, 2009; Tracy, 1997; Morita, 2000; Kobayashi, 2006; Zappa-Hollman, 2007b; Duff & Kobayashi, 2010). The participants in Language Socialization events are typically termed *experts* and *novices*. For novices, "participation in communicative practices is promoted but not determined by a legacy of socially and culturally informed persons, artifacts, and features of the built environment" (Schieffelin, 2011, p. 4). As it developed, Language Socialization Theory became concerned with the active role of the novice in constructing the conceptual and linguistic ideas required to participate fully in their speech community (Heath, 1983; Paugh, 2012a, b), as well as the expert members' ability to provide the required space and opportunity to do so (Ochs & Schieffelin, 1986; Blum-Kulka, 2017). While language socialization is inspired by the psychological theory of socialization (Ochs & Schieffelin, 2014), it diverges from early socialization research by avoiding the determinism implied by a

framework that posits the expert as the agent of socialization to the learner' recipient. Instead, the novice is understood to be the genesis of culture, which is nurtured by society (Sapir, 1924; 1933). Language socialization studies are rare that describe socialization to very complex language forms (Duff, 2008a): work has been done describing socialization to use physics terminology (Jacoby & Gonzales, 1991; Jacoby, 1998), law jargon (Philips, 1982), and medical notes (Hobbs, 2004), but these represent only a very small sample of the speech communities of this type. These analyses tend to focus on speech patterns and cultural identities that are considered important across speech communities sharing a particular quality, such as hair dressers (Jacobs-Huey, 2003) or lawyers (Philips, 1982). Another set of law students socialized into the practices of Socratic dialogue, courtroom simulation, and recontextualizing legal precedent to support new arguments as they moved toward the goal of joining professional law communities (Mertz, 1996, 1998, 2007). In an American hospital, new doctors learned conventions for notes and abbreviations for patient reports via language socialization (Hobbs, 2004).

Socialization events between participants can be analyzed as taking the form of modeling, feedback, and uptake (Duff, 2010). An expert models linguistic, paralinguistic, and nonlinguistic forms in appropriate contexts. A novice provides feedback by establishing *stances*, or displays of an attitude toward a proposition (Biber & Finegan, 1989; Ochs & Schieffelin, 1989). Lastly, they show that they have taken up the relevant forms by using them appropriately. The expert-novice relationship necessarily contains an element of asymmetry in knowledge and power, wherein the informed party is more likely to hold ratified knowledge and will not grant social advantage to possessors of unratified knowledge (Ochs & Schieffelin, 2011; 2014). However, uptake does not always perfectly match the expert's model, as novices have agency in

supporting, rejecting, or modifying the forms according to their judgment (Kulick & Schieffelin, 2004; Morita, 2009; Goodwin & Kyratzis, 2012; Rogoff et al., 2014 Referring to "experts" and "novices" at all has been problematized (Li, 2000; Duff, 2008a). Despite the asymmetries in power that differentiate them from experts, novices exercise significant agency as creators of the social reality in which they participate (Bourdieu, 1977; 1990). Experts, artifacts, and features of the society in place are tools by which all participants create this reality (Piaget, 1952; Cole & Cole, 1989). Because of this, full participation is not necessarily measured by how well novice behavior matches that of the expert, and novice feedback can affect an expert's view of what is appropriate (Rogoff, 1990). Even refusal to participate in a particular act can be considered a type of socialization (Duff, 2002; Morita, 2004). Novice preferences for non-expert-like language use can be explained by the idea that language socialization is characterized by the dual importance of predictability and spontaneity (Ochs & Schieffelin, 2014). While repetition is necessary for novices to acquire interactional routines (Schegloff, 1987), the end stage of acquiring those routines is the ability to improvise (Duranti & Black, 2014).

Co-construction (Jacoby & Ochs, 1995) is a joint activity of creation: participants can create culturally meaningful realities between themselves that might not have been possible in isolation. Early observational studies noted that participants' identities and perspectives can intermingle over the course of a social interaction (Leontyev, 1981). This intermingling of identities and perspectives results in the participants' learning from themselves and one another simultaneously. Co-construction does not necessarily have to be affiliative or supportive, however: participants in conflict may in fact co-construct their social reality while in opposition to each other (Jacoby & Ochs, 1995). There is a significant open space for peer socialization that describes adult peer socialization practices, such as Jacoby and Gonzalez's (1982) study of

Physics graduate students practicing conference presentations, who socialized one another through conference talk rehearsals using critiquing comments, focusing particularly on presentation timing and slide design, relatively low-risk items about which to venture critiques. Philips (1982) observed law students as they socialized into a community that uses a specific legal cant style, concluding that the practice of dividing people by level allowed peers to spend significant time practicing with one another, contributing to their mastery of that highly specialized style. While earlier studies of Language Socialization observed groups of relatively homogeneous identities, the flexibility of this model allows for the complex interplay of diverse cultures when they meet in a community (Riley, 2008). The treatment of socialization events as discrete and observable objects creates an opportunity to observe socialization even within complex and heterogeneous groups. Alongside language development there are also intervening variables of social identity, gender, and class which shape the ways learners' abilities are perceived by others. (Willett, 1995; Baquedano-Lopez & Kattan 2008). While it is understood as a basic tenet of Language Socialization Theory that the linguistic and social acts that constitute it contribute to a common understanding of a shared set of values, researchers must not presume those values, nor can they consider that common understanding to be absolute or permanent (Riley, 2008).

Because language socialization is understood as repeated encounters dedicated to the construction of knowledge about and active participation in a community (Ochs, 2000), practices involved in Language Socialization can be zoomed out to draw conclusions about larger cultural structures (Duff, 1995; 1996; 2002; Ochs, 2000; Riley, 2008). Talk between novices and experts is seen as a set of cultural arrangements achieved through participation (Ochs, 2000), which means that each individual action can be examined as indicative of larger ideologies as to what is

normal and acceptable. In the context of institutions such as schools and workplaces, which have complex relationships to the larger communities in which they find themselves, language socialization practices constitute those relationships (Dunn 1999, He 2001, Jacobs-Huey 1999, Jacoby 1998, Rymes 2001). Classroom routines provide insight into the socialization of the student to and through language, as well as to the structure of the institution (Anderson, 1995; Baquedano-López, 1997, 2000; Cook, 1999; Duff, 1995; Willett, 1995). As the goal of socialization is to become an expert in the use of language -- linguistic features as well as those social categories that language communicates -- in socially constructed communities, it is important to discover the ways in which Language Socialization Theory conceptualizes the social.

Identities, Stances, Ideologies

Research observing language socialization in action has suggested that there are several social factors that affect the type and amount of learning that takes place: orientation to the community, negotiation of power, and identities are among these factors (Duff, 2010). Participants know that there are social consequences to the choices they make in terms of language use (Blom & Gumperz, 1972; Gumperz, 1982a, 1982b; Heller, 1988; Sacks, Schegloff & Jefferson, 1974; Woolard, 1985) and that language use is a form of social action (Austin, 1962; Searle, 1969, 1975). Rather than a stable, immutable "self," *identities* are understood in sociolinguistics to be cultural concepts that people interact with through language, gesture, and other communicative acts (Butler, 1990; Ochs, 1993: Eckert & McConnell-Ginet, 1992; 2013). The relationship of language to identity is mediated by speakers' understanding of conventions (Ochs, 1993). In a study of high school students, Eckert and McConnell-Ginet (1995) discovered

that children construct and *perform* identities corresponding to the available categories they know to be culturally salient, despite no obvious essential traits that would indicate ahead of time which personae they would take on. Stances and ideologies can change as adults are socialized into new contexts: in an observation of defense analysts, Cohn (1987) observed her own change in attitude toward violence as she participated in conversation that constructed violent events in sanitized or even positive terms. Rather than defining identities as the source of speech and action, sociolinguists prefer to reverse this association. Because these actions can be understood as producing identities, they are referred to as *performativity* (Butler, 1990; Mendoza-Denton, 2008). Performing an identity here does not mean pretending, instead it is a way of bringing something into being through the performance (Bauman, 1986; Hymes, 1975; Butler, 1990; Bucholtz & Hall, 2004). In this way, we can look for patterns while also recognizing that human cognition is so complex that our self-definition is frequently nonlinear and contradictory. Considering identities as an active process rather than an attribute -- essential or not -- implicates practices, not people, as the fundamental units of identities (Eckert & McConnell-Ginet, 1992; Ochs, 1993).

Some engineering educators want to consider identity as constructed and negotiated through interactions (Burke & Stets, 2009; Downey & Lucena, 2003; Tonso, 2006; Wenger, 1998). Following theorists in related fields, they began to take student background as an active focus of study rather than a stable input variable (Kanny et al, 2014). As such, psychological factors evolved from stable personality ideas to contextual, active concepts like sense of belonging (Kanny et al., 2014). Those who study this process argue that while participating in interactions, individuals evaluate them and establish stances to identities they meet: it may be that computing identity development is an extension of science identity (Pozzer & Jackson,

2015). Theorists who write about the relationship of language to other socially mediated behaviors analyze individual acts of performing social identities as consisting of acts of *positioning* (Davies & Harré, 1990). Social Positioning Theory, a framework for understanding identities, performativity, and positioning, has been employed to describe how identities affect language socialization processes (e.g., Ochs, 1993, 1996; Paechter, 2003; Mendoza-Denton, 2008). The process by which novices learn the practices required for full participation is also well-suited to discussions of other social structures such as gender and race because of its applications as a description of identity formation (Paechter, 2003), an area CS education is interested in exploring but has not yet done (Rodriguez & Lehman, 2017). In educational contexts, learners are reflexively and interactively positioned as capable or incapable, worthy or unworthy, and their claims to legitimate membership will be assessed (Duff, 2010). People engage in *interactive positioning*, the practice of casting one's interlocutor in a particular light, a kind of performance on another's behalf (Davies & Harré, 1990). They also *reflexively position* themselves, applying an identity to themselves through literal messaging, metaphors, and backgrounded messages that rely on presuppositions (Davies & Harré, 1990; Eckert & McConnell-Ginet, 2013). Presuppositions are made possible by conceptual baggage, the conventional associations between categories and discourses established in communities. Positioning can be accomplished through linguistic choices – wording, grammar, pitch, accent, and more – or through other practices such as clothing, gesture, and posture. It is also important to remember that academic socialization is multi-directional, allowing for experts to be socialized by their junior associates or peers (Duff, 1995; Talmy, 2008). For example, Talmy (2008) observed students' resistance to teacher attempts to position them, which resulted in the teacher, not the students, accommodating his stances.

Social identities are created and re-created through repeated performances (Bauman, 1986; Hymes, 1975; Butler, 1990; Bucholtz & Hall, 2004), some of which are achieved by displaying and ratifying *stances* (Ochs, 1993). A stance can be an *epistemic* attitude, expressed in terms of degree of certainty (Chafe & Nichols, 1986) or an *affective* attitude, an expression of emotion of varying intensity and type (Besnier, 1990; Ochs & Schieffelin, 1989). Taking up stances with respect to a peer, or something toward which one's peer has claimed a stance, can reference the group's notion of appropriate behavior (Du Bois, 2007). Stances are posited as early as infancy: babies show recognition or use of facial and prosodic markers of affect (Halliday, 1975; Campos & Stenberg, 1981; Crittenden, 1986). Babies also look to their mothers for information on what stances they should take, an act called *social referencing* (Campos & Stenberg, 1981; Stoufe, et al., 1984), which applies to any event in which one participant seeks out the stance of another. Like other language practices, academic discourse is constituted by social processes and stances (Duff, 2010) that involve dialogically constructed emotional investment and power dynamics (Molle & Prior, 2008; Duff, 2010). To agree with an interlocutor's stance is to *ratify* it, an important part of co-construction. This ongoing negotiation allows for variation between and within social categories (Ochs, 1993) so that there are several different types of person who can position themselves to any identity. Peers are able to take stances using methods such as *assessment, membership categorization,* and *dialogic voicing*. Peers can assess one another explicitly (Goodwin and Goodwin, 1987; Goodwin, 2006) or through gossip (Morgan, 2002; Evaldsson, 2002; Mendoza-Denton, 2008) based on their ability to behave and speak appropriately. Membership categorization allows peers to position themselves and others as members or not by choosing whether to ratify their own or peers' knowledge (Goodenough, 1965; Sacks, 1972). Dialogic voicing, quoting the real or imagined

speech of others, also constitutes a stance (Bakhtin, 1981; Goodwin, 1990). Language Socialization theorists take a social constructive approach (Gumperz, 1982), asking, "What kind of social identity is a person attempting to construct in performing this kind of verbal act or in verbally expressing this kind of stance?" (Ochs, 1993, p. 296). A speaker's failure to establish a socially recognizable or appropriate identity may not be a failure of linguistic competence but a lack of understanding as to how this particular community has conventionalized identities (Ochs, 1993).

The connection of identities to cultural concepts is solidified through four semiotic processes established in linguistic anthropology: these are practice, indexicality, ideology, and performance (Bucholtz & Hall, 2004). *Practice* is defined as habitual social activity (Bucholtz & Hall, 2004); this is a concept taken from Bordieu (1977) – it is through practice that we make ourselves as we engage with communities of practice (Lave & Wenger, 1991b; Wenger, 1998). *Indexicality* is the ways in which practices and stances point to larger sociocultural concepts (Silverstein, 1989; Ochs, 1986; Bucholtz & Hall, 2004). Linguistic structures become associated with social categories by repeated affiliation with those categories. For example, people in many contexts can use mitigation and increased variation in pitch to index femininity (Bucholtz & Hall, 2004). *Ideologies* are organizations of cultural beliefs, practices, and power relations (Blommaert, 1999; Kroskrity, 2000b; Schieffelin, Woolard, & Kroskrity, 1998; Bucholtz & Hall, 2004). These beliefs, practices, and relations are also socially negotiated, but they are often so entrenched that they are considered by members to be common sense. Ideologies come into being through a process of *erasure*, the elimination of details that are inconsistent with a belief, *iconization,* pairing a linguistic or other feature to a particular identity group, or *essentialism,* the

creation of a naturalized pseudo-logical link between the linguistic and the social (Irvine & Gal, 2000).

Many authors have written about the role of ideologies in language socialization at schools, especially in terms of the structuring of school practices (Fader, 2001; Field, 2001; Jaffe, 2001; Baquedano-Lopez & Kattan 2008). Ideologies concerning the professional academic identity tend to claim that legitimate members practice hypothesizing, claiming, instructing, assessing, and they take stances of objectivity, knowledgeability, and intellectual flexibility (Ochs, 1993). These four processes contribute to the mutually agreed-upon associations members of a community form between identities and their beliefs about the culture. CS education researchers are aware of the challenges students. While the field of CS overwhelmingly employs men now, the ideology that it is a masculine field is socially constructed: over time, programming changed from a woman's clerical job to a creative and challenging man's job that requires analytical thinking, scientific activities, and intellectualism (Holland, 1997; Smart, Feldman, & Ethington, 2000; Ensmenger, 2010). Computer Science education contexts took on these ideologies, emphasizing individualism (Waite et al., 2004), competitive practices (Barker & Garvin-Doxas, 2004), and reluctance to express emotion (Barker & Garvin-Doxas, 2004). Rather than focus on the real-world applications of programming, many courses tend to focus on theoretical and abstract topics (Cech, 2014). Since then, CS thought processes developed from a white middle-class male perspective, and therefore represents their values (Faulkner, 2001; Björkman, 2005). Those who do not identify as white men struggle to advance because of the norms enforced by those who are already full members, norms that those full members may not even recognize as originating from racial or gendered identities (Rasmussen & Håpnes, 1991).

In the space of language socialization into a work or school community, learning is characterized by complex relationships to the larger ideologies and discourses in which it is situated (Dunn 1999, He 2001, Jacobs-Huey 1999, Jacoby 1998, Rymes 2001; Roberts, 2010). Those who enter new language contexts tend to experience shifts in their ideologies and identities (Gordon, 2004). Students in classrooms are often socialized into and through practices of showing respect to teachers and to the subject matter, self-control, and decorum: they learn ideologies of respect as well as the linguistic forms associated with showing it (Howard & Lo, 2009; Talmy, 2009). In elementary school classrooms, classroom discourse can provide clear evidence of other types of ideologies, too, such as the need for autonomous and independent academic work versus collaboration and shared knowledge production and ownership (e.g., Toohey, 1998). This applies to higher education and workplaces as well; for example, Kim & Duff (2012) found that Korean-Canadian female students were socialized into various beliefs/ideologies about language, including gender- and language-related identities. For example, students who participated in this study reported hearing from slightly older peers that fluent English use would be impossible for them as non-native speakers, but that speaking only Korean in social contexts was pathetic. In English-medium contexts, language ideologies tend to consider white native speakers to be the only legitimate ratifiers of language use (Norton, 2000). In a study of Lao men and women in the United States, Gordon (2004) argues that because of pervasive ideologies that delegitimize the language abilities of certain members, the classroom is an important place to discuss the linguistic, pragmatic, and identity challenges newcomers to the United States will find outside the classroom. This is especially relevant to researchers such as Guardado (2009), who found that Spanish-speaking families in Vancouver often accidentally reproduced dominant ideologies even as they attempted to challenge them through practices that

felt "natural" to them, such as switching to English for activities that carried institutional import. Certain aspects of academic discourse socialization research have been problematized on the basis that they can presume that experts are good, competent socializers, or have even fully mastered the requisite practices themselves, or that the biggest challenge for students in academia is formal technical or academic written discourse rather than other more interpersonal forms of discourse and communication found in class discussions or other academic interactions (Duff, 2004, 2007b, 2010; Bunch, 2009). In extremely prestigious academic communities, it has been suggested that the inaccessibility of academic discourse is deliberate as it secures the experts' positions (Bourdieu, Passeron, & Saint Martin, 1994).

Another issue concerning academic discourse socialization is the potential mismatch between its expression in schools as opposed to in the workplaces those schools are meant to prepare students to enter. The workplace involves interactional requirements of three types: the corporate or institutional type, the professional type, and the social or personal type, all three of which are required to demonstrate that a learner is becoming a full participant in the community of practice (Roberts & Sarangi, 1999). In many professional contexts, *corporate discourses* around what it takes to thrive in industry involve expectations that successful members will be empirical, deductive, individualistic, and egalitarian (Mawer, 1999; Iedema, 2003; Poncini, 2003; Jack, 2009). The *professional discourses* concerning how people should behave at work vary depending on the space, but experts socializing an engineering team observed by Vickers (2007) employed observation, scaffolding, ridicule, and opportunities to speak during the design process. Students in higher education must learn to participate effectively in practices that involve being assessed by instructors (Mehan, 1979). Lastly, *personal/social discourses* in higher education or professional communities of practice involve what identities have a place there;

these discourses reinforce acceptable ways of doing politeness, humor, or other pragmatic choices at work (Katz, 2000; Holmes, 2005a; Holmes & Stubbe, 2003; Daly, Holmes, Newton, & Stubbe, 2004; Gunnarsson, 2009). Because much language socialization research requires long-term ethnographic observation of natural speech, there is little research on adults in workplace or educational contexts, not least due to logistical challenges (Duff, 2010). Most workplaces require appropriate and competent communicative practices (Matthewman, 1996); the practices actually present in professional settings require collaboration; however, there appear to be pervasive discourses of high academic or professional achievers as individualistic workers (Heath, 1989). This marks a problematic disconnect between classroom socialization and workplace practices that privileges white middle class ways of doing, which can leave non-white, non-middle-class students feeling that they are unable to understand the social and cultural references that teachers use to communicate new content. Another challenge for students is that several researchers report a disconnect between classroom practices and the workplace practice they are meant to be preparing students to participate in: for example, Canadian nurses in school learned how to write nursing reports and care plans, but when they arrived they discovered that their practices were considered wasteful, and had to learn to write them using different style and content once they arrived at their hospitals (Parks & Maguire, 1999; Parks, 2001). Nursing home employees who learned to participate in technical oral discourse with colleagues and educators discovered an extremely different set of expectations in the workplace due to patient communication difficulties, such as simplified language, slower rates of speech, reducing jargon, and incorporating nonverbal strategies (Duff, Wong, & Early, 2000). In CS workplaces, sociolinguistic studies have posited that the communicative requirements of Silicon Valley communities are that employees be able to volunteer personal opinions and publicly demonstrate

their technical abilities, a skill that may prove challenging to newcomers (Hull, 1997; Katz, 2000). There are likely many other identities, discourses, and ideologies at play in CS education, but these have yet to be explored (Rodriguez & Lehman, 2017).

Language Socialization and Face Threat

The interactional events associated with socialization can be described as negotiations of *face*, a concept in Pragmatics. Pragmatics is a subfield of Linguistics that is concerned with making systematic inferences about the intention behind an utterance and its effect on the hearer on the basis of its context and the speaker's choice of phrasing (Birner, 2013). To be able to make claims about these internally experienced and highly subjective phenomena, a range of methods have been considered. Foundational essays and books in this field adopted a constructivist approach (Grice 1957, 1969, 1975; Austin, 1962; Brown & Levinson, 1978; Searle, 1969), which revolve around a Model Person who natively and fluently speaks the language(s) in question, and who possesses rationality (Brown & Levinson, 1978). This Model Person is placed in imaginary contexts and the question of whether something they said would be grammatical, logical, or socially acceptable is considered. Others prefer to rely on interviews with informants or experimental methods (Gibbs, 1986; Airenti et al., 1993). Later studies make use of corpora to define the range of preferred utterance types (Kerssen-Griep et al., 2008; Brummernhenrich & Jucks, 2016).

Several foundational principles of Pragmatics continue to be in use in the twenty-first century (Archer et al., 2012; Birner, 2013). Austin (1962) and Searle (1969), among others, developed the initial framework around the real-world effects of speech. Speech Act Theory (Austin, 1962; Searle, 1969) builds on the concept of forces by describing some of the types of

events that speech can bring about. A speech act is not only a statement but is also an action: one can cause a change in another or in oneself simply through words or gestures. For example, if a speaker (S) asks a hearer (H) to "pass the salt", S manifests an expectation that H will comply, thereby performing an action. Searle (1975) set forth five types of speech acts. These are assertives (statements that commit the speaker to the truth of an expressed proposition, such as a creed), directives (statements that cause the hearer to do some action, such as a request or command), commissives (statements that commit the speaker to a future action, such as a promise or a threat), expressives (statements that report the speaker's attitude toward some proposition, such as a congratulations or an excuse), and declaratives (statements that change the institutional status of some party, such as a marriage or a sentencing). A speech act can be said to be composed of three forces: the locutionary, the illocutionary, and the perlocutionary force (Austin, 1962). The locutionary force of an utterance is its literal, semantic meaning. The illocutionary force is the intention behind the utterance; in the case of a speech act, this is the plan S has enacted to create an event such as a request, a warning, an offer, or some other action. The perlocutionary force is the actual effect on the hearer. In the case of "pass the salt," the perlocutionary force is usually that H passes the salt to S. By striating speech acts into three levels, Austin allowed analysis of each one individually, making it possible to determine conditions for the successful execution of each.

Searle (1979) further developed Speech Act analysis by drawing a causal connection between changing the locutionary force of an utterance and eliciting a different perlocutionary force in the hearer. Speakers appear to risk something by being too blunt, and choosing a more indirect wording allows them to maintain the illocutionary force of a speech act while avoiding potentially negative perlocutionary forces. In the case of a directive, a direct imperative ("Shut

the door," e.g.) may not be the preferred way forward, as such a blunt command could be considered an affront. In fact, the method the speaker uses to change the wording reveals something about the context. A speaker might instead choose to ask after H's ability to perform the action, declare the existence of a problem, expecting that H will choose to perform the action, or ask whether there are good reasons not to perform the action. Which one is best depends on several social and contextual conditions, the definition and analysis of which became a central pursuit of research in politeness.

H.P. Grice's (1975) interest in conversation and the reciprocal relationship between speaker and hearer led to a formalization of the dimensions along which an utterance can be considered successful, and the enumeration of these dimensions provided a framework for analyzing off-record indirectness. Grice argued that speakers prefer to be understood and will adhere to a Cooperative Principle as a guiding force in their choices, and that hearers will assume the same, leading them to a successful interpretation of the speaker's utterance. The Cooperative Principle consists of four maxims: these are Quantity, Quality, Relation, and Manner. To comply with the Maxim of Quantity, a speaker must be as informative as required and must avoid providing more information than is required. To comply with the Maxim of Quality, speakers must avoid saying what they believe to be false, nor should they claim a truth for which they lack adequate evidence. The Maxim of Relation requires speakers to speak on topics that are relevant either to the preceding utterances or to the speakers' surroundings. The Maxim of Manner requires a speaker to avoid obscure, ambiguous, overly wordy or illogically ordered utterances. The Cooperative Principle states that speakers will choose to follow these four maxims, and if they flout one or more of them, the fact of this violation has in itself a context-specific meaning. The hearer, then, is prompted to make a conversational implicature, an inference as to the reason

the speaker might have violated a maxim. This behavior provides a basis for claiming that conventionalized indirectness (phrases that everyone associates with an illocutionary speech act rather than a literal meaning) becomes part of language to maximize tact but minimize confusion.

Language Socialization studies note that experts use humor, teasing, and small talk, which can help accelerate socialization (Holmes, 2005b Holmes & Schnurr, 2005; Mak et al., 2012). These tools are considered necessary in order to increase students' comfort and confidence, which suggests that language socialization is characterized by several types of *face threat.* In the course of performing identities as members of varying levels of participation within a particular speech community, various interpersonal challenges present themselves. These challenges can be analyzed as threats to *positive* and *negative face.* Politeness Theory, a framework within Pragmatics, has been partly concerned with formalizing the concept of face as a representation of the perlocutionary force of speech acts. In other words, Politeness Theory provides a way of analyzing how speakers manage the effect of their utterances on others by adjusting how, or even if, they speak. Brown and Levinson (1978) split face across two dimensions: Positive Face and Negative Face. Positive Face is the sense that one's self image is appreciated and approved of by others. Negative Face is a sense of autonomy and freedom from imposition. Cognitive studies of learners support Brown & Levinson's initial formulation of two key types of face needs: a sense of competence, or positive face, and a sense of autonomy, or negative face (Lim & Bowers, 1991; Ryan & Deci, 2000; Kerssen-Griep et al., 2003).

Language socialization into new academic or professional communities connects with *face* and *facework* because pragmatic and social skills are learned informally and have a direct effect on workplace efficacy (Tracy, 1997; Li, 2000; Duff, Wong, & Early, 2000; Duff, 2009,

2010; Roberts, 2010). While descriptions of face generally appeal to a sense of self, it was not necessarily formally combined with social psychological concepts of identity until relatively recently (Spencer-Oatey, 2007; Scollon & Scollon, 2011). Identity can be understood in discrete moments of performance, as categories and types that are brought into being by negotiation (Scollon & Scollon, 2011). Speakers can claim identities for themselves by enacting traits that are commonly associated with those categories, or they can assign identities to others by treating them as members thereof. It is through facework that people construct their identities. Translating face from a possession to a process allowed Face Theory to apply to interactional models of identity (Arundale, 2006; Spencer-Oatey, 2007), a development that has allowed increased versatility and cooperation with theories of social psychology and cognition. Socialization activities that require oral discourses involve potential for serious conflict, tensions, and loss of face (Tracy, 1997).

Many important speech events associated with higher education involve considerable challenges to each participant's desire to affirm negative and positive face in their interlocutors, particularly in the case of teacher-student dyads. Assertives, advice, agreement, disagreement, and directives are the speech act types of particular interest in the case of teacher-student and student-student interaction. Instructors provide assertives and pieces of advice as they convey course information to the students. When students submit their interpretation of the course material or propose solutions to assignments, instructors must agree or disagree in order to convey the correct answer. During classroom management and while checking student work, they must issue directives. Students talking to students will be required to perform many of the same actions, but they may employ different strategies to accomplish that goal. Recent experimental attempts to test the effects of instructors' politeness strategies have taken

questionnaire-based and ethnographic approaches. Some experimental studies that operationalize Brown & Levinson's (1978) politeness strategies take as their dependent variable a third-party witness's opinion of the instructor or learning experience rather than the effect on the student's success (Kerssen-Griep et al., 2008; Brummernhenrich & Jucks, 2016). Others use questionnaire-based measures of intrinsic student motivation and objective measures of classroom involvement to demonstrate an effect of students' recollections of individual facework events (Kerssen-Griep, 2001; Kerssen-Griep et al., 2003). An ethnographic study of writing tutors concluded that at the beginning of their relationship, instructors use negative politeness strategies to maintain a collaborative peer identity and an authoritative instructor identity, and as they decrease in social distance, tutors rely more on positive politeness strategies instead (Bell et al., 2009).

Very few studies have been done that examine the potential pitfalls of facework and politeness strategies in the learning context, but Person et al. (1995) found in a corpus-based study that student outcomes were sometimes limited by instructors' overuse of politeness strategies during conversations. Because face-threatening acts are highly likely in the course of negotiating answers and locating students' knowledge deficits, when a tutor uses a politeness strategy to reduce the impact of such an FTA, the educational benefit of that FTA is also reduced. Further discussion concerning how facework and FTAs will be applied to the data collected in this study appears in Chapter 3.

Learning and Cognition: Legitimate Peripheral Participation

Research investigating Computer Science pedagogy has not involved Linguistics because the authors of publications in the field of CS Education are generally either instructors

themselves, writing about the methods they use and have found effective, or they are psychology researchers, who conduct experimental studies to test the effects of certain factors on student scores. The contents of CS courses include programming, algorithmic thinking, information management, as well as design, social impact, and cultural context of both software and hardware (Sheldon & Turbak, 2008; Tedre et al., 2018). Some researchers believe it is impossible to turn novices into experts in a four-year course and that the best educators can hope for is to foster basic competency (Winslow, 1996). Pedagogical theories often are not incorporated into the design and execution of CS courses (Winslow, 1996; Phillips, 2005; Clear, 2006). Rather than discuss theories of learning and the mind, publications interested in Computer Science Education are more centered around which topics to include (Sheldon & Turbak, 2008; Clear et al., 2017) and testing the effects of common techniques on student performance (Nagappan et al, 2003; Mendes et al., 2005; Mendes et al, 2006; Shaochun & Rajlich, 2006; Umapathy & Ritzhaupt, 2017). In these studies, student performance is measured through exam scores, homework grades, or course grades rather than linguistic behavior. Those interested in encouraging students to stay in CS programs have found that student experiences in the first year shape their ability to identify as computer scientists, which affects their intent to continue (Peters, 2014; Peters et al., 2014; Peters & Pears, 2012, 2013). They also focus on student backgrounds, family influences, structural barriers in K-12 education, psychological factors, and perceptions of STEM (Kanny et al., 2014).

Most research concerning how to maximize computer science students' academic outcomes tends to focus on learning by doing as the best mechanism for developing basic programming abilities in university students (Winslow, 1996; Sheldon & Turbak, 2008). Learning by doing is a generalized term used to describe the classroom events that involve

students creating solutions to problems posed by instructors or instructional materials, and executing those solutions by writing code (UC Davis). These methods are devised primarily by instructors who have field-tested their ideas and stand by them on the basis that they worked well in practice, but they also have several elements in common with theories of learning within and without the field of Computer Science Education. The aim of having students learn by doing may have some roots in constructivism (Ben-Ari, 2001; Boudourides, 2003), a framework that posits that the cognitive process of learning is an act of constructing meaning from experience (Duffy & Jonassen, 1992). Constructivism has exerted an increasing influence on education at all levels of schooling in North America and Europe (Di Vesta, 1987; Duffy & Jonassen 1992; Steffe & Gale, 1995; Hiebert et al., 1996; Ben-Ari, 2001; Boudourides, 2003; English, 2003; Verenikina, 2004). This approach defines development and learning as interrelated, a process of interaction with the world in which we develop epistemologies by moving within the world and negotiating relationships with other people. Constructivist methods tend to move away from teaching a concept and then asking students to apply it; teachers are instead encouraged to use the application as a context for learning a concept (Hiebert et al, 1996). With guidance from teachers, this approach takes advantage of emerging understandings of cognitive development that suggests that context is an integral part of learning and that learning by doing is a way to maximize the depth of acquisition of a skill (Duffy & Jonassen, 1992). This guidance takes the form of co-construction of knowledge wherein teachers ask students to participate in the development of tools that they will then use to interpret new information as it is introduced. Tasks that might be slightly too complicated for a novice can be interactionally achieved if they coordinate with a more expert member. This perspective has several important parallels to Language Socialization.

Social constructivism relies on the same foundation as constructivism and gives an opportunity to apply Language Socialization theories to constructivist teaching practices (Atwater, 1996; McGroarty, 1998). Where earlier educational theories took as their aim the transmission of decontextualized knowledge and skills, followed by practice applying that knowledge to different contexts, later cognition studies concluded that there is no such thing as decontextualized learning (Perret-Clermont, Perret, & Bell, 1991). It builds from work by Piaget and others that investigates the complex relationship between development and learning (Vygostky, 1978). In this framework, cognition is considered a response to a situation (Resnick, 199), and learning is engagement in human behavior whether in concert or in conflict with others (Lave, 1993). This interplay of social constructivism and cognition can be analyzed using concepts of *habitus* and *practice* (Bourdieu, 1977b), the creation of knowledge through living and acting in social contexts. Language socialization studies employing these concepts investigate how learners' lived experiences form their knowledge of the community they are joining (e.g., Toohey, 2000, Norton, 2000; Miller 2003). Imagining learning as a process of becoming integrated with a real or imagined community assumes that humans are, above all, social, that knowledge is demonstrable competence in valued enterprises, which can be anything from sewing to programming to being a boy, that knowing is participating in the pursuit of those enterprises, and that meaning is the ability to experience the world as meaningful (Wenger, 1998). This is the goal of learning. By participating in and acquiring competence in new enterprises, the learner begins to construct new identities, which leads to a sense of belonging in the community.

A relevant theory to the pursuit of applying Language Socialization to research on CS education is *situated learning*. Situated learning is sometimes presented as an alternative to

constructivism (Ben-Ari, 2004) but it relies on similar premises and has similar goals. This endeavor is the result of an emerging theory that "there is reason to suspect that what we call cognition is in fact a complex social phenomenon" (Lave, 1988). This framework created a link between cognitive psychology and anthropology that forced theories of learning to address the multifaceted, personal, and social nature of creating knowledge. Lave recommended that the fostering of communities of practice is ideal for maximizing knowledge building and retention because it creates living knowledge that is renegotiated and updated automatically through social processes, making it more flexible and more durable than knowledge stored in a database. Lave and Wenger came to consider situated learning to be a reverse of the actual state of affairs (Lave & Wenger, 1991a). Situated learning is the end to which practice is the means, while *legitimate peripheral participation* positions social practice as a phenomenon by which learners generate the world, of which learning is but one type. This is the process by which newcomers move toward full membership of a community – rather than defining situated learning as "learning by doing" exclusively, legitimate peripheral participation arose as a solution to the confusion around what constitutes active learning and apprenticeship. Novices participate in situated learning through a variety of practices including observation, discussion, and "doing," all of which are, in practice, legitimate means of transmitting knowledge. In a classroom, legitimate peripheral participation can look like a classroom discussion, an exam, collaborative problem sets, or any number of other school practices. Individual orientation – meaning, orientation to the community - can also have a significant effect on the speed at which a novice acquires new linguistic and communicative skills; those who are most motivated to be socially active in their class will move more quickly than those who are less motivated (Fillmore, 1979; Duff, 2010).

Vygotsky's (1978) *zone of proximal development* can also be a useful tool for observing the kind of learning that happens through interaction. The zone of proximal development is the distance between a task one can do with help as opposed to alone. It does not posit that information always passes from teacher to learner in a direct path; instead, learning is a gradual increase in expert-like expressions of participation. It takes the world as socially constituted, meaning that each learner generates knowledge within themselves through social interaction with experts and other novices. While it is rare for Computer Science education researchers to mention legitimate peripheral participation and the zone of proximal development, the collaborative activity *pair programming* is mentioned as an event that is characterized by its presence (Salleh et al., 2010). Students who participate in pair programming are able to reach higher intellectual performance than those who do not, suggesting that resolving challenges collaboratively with no expert present may increase their chances of finding themselves in the zone of proximal development.

Collaboration and Pair Programming

Beyond learning-by-doing or simply emphasizing the importance of practical experience, empirical research into effective Computer Science (CS) teaching practices suggest that collaboration is beneficial to students (McDowell et al., 2002; Hancock, 2004; Barker, et al., 2009; Alvarado & Dodds, 2010). Working with other students has been found to promote persistence and encourages students to practice *adaptive helpseeking* (Newman, 1994; Israel et al., 2016). Adaptive helpseeking describes the act of soliciting help from a peer in ways that most build competence and confidence. Students who do "adaptive helpseeking" tend to construct questions such that they must demonstrate more of their own knowledge of the subject,

and they are more likely to ensure that they have reached a satisfactory conclusion than those who do not. This includes asking questions using specific and relevant terminology, describing the problem with accuracy, and asking follow-up questions to ensure an answer is reached. It is also suggested that collaborative problem-solving promotes proactive resolution of uncertainty, which gives students a greater sense of ownership over the knowledge they generate, and the activity also appears to increase the degree to which students are socialized into programmer culture (Israel et al., 2016), an important requirement for developing a sense of belonging while studying and working in CS.

As an extension of the preference for learning by doing, a popular professional training practice called "pair programming" (Williams, 2000) has been tested in classroom environments to positive results (Williams et al, 2000; Werner et al, 2004; Shaochun & Rajlich, 2006; Salleh et al, 2010, for example). Pair programming is an activity in which two individuals write programs on a shared computer (Williams et al., 2000; Williams & Kessler, 2002). One partner, the "driver," types into the development environment while the other, the "navigator," dictates the design and implementation of the solutions. In some cases, the navigator does all of the talking and the driver merely types what the navigator says, but more often than not there is significant conversation between the two participants, and the two reach conclusions together.

Experiments incorporating pair programming into classroom activities have yielded several positive reports. Psychology and Education research suggests that pair programming in the classroom can undo some of the negative effects of heterogeneity within a classroom (Klopp et al., 2017). Some pair programming experiments suggest that it improves student affect (Cockburn & Williams, 2000; Williams & Kessler, 2000; Williams et al., 2003; McDowell et al., 2003; Mendes et al., 2005) and confidence (McDowell, et al., 2003; Hanks et al., 2004),

particularly in women (Berenson et al., 2004), while others do not find that affective measures are improved by its implementation (Umapathy & Ritzhaupt, 2017). Introducing pair programming into a beginning CS course improves course completion rates (McDowell et al., 2003; Nagappan et al., 2003) and encourages students to find intrinsic motivations to pursue CS, which increases perseverance (Klopp et al., 2017). As discussed in Chapter One, several questionnaire-based studies have concluded that women tend to report more extrinsic motivation for pursuing CS, such as opportunities for a lucrative career, while men tend to report that they study CS because they love programming as an activity (Margolis & Fisher, 2002; Leder & Vale, 2004). However, Doube & Lang (2011) found that men and women enrolled in a CS major both demonstrated high intrinsic and extrinsic goal orientation. By increasing intrinsic motivation through the use of activities that increase positive emotions toward CS, women may be especially impacted by the inclusion of pair programming. Pair programming appears to improve student scores on measures of competency such as researchers' assessments of submission quality (Salleh et al., 2010; Klopp et al., 2017). It has also been associated with increases in homework submission rates (Hanks et al., 2004) and test scores (Nagappan et al., 2003; Mendes et al., 2005; Mendes et al., 2006; Salleh et al., 2010). While pair programming often causes students to take more time per problem (Cockburn & Williams, 2000), it also improves design quality (Cockburn & Williams, 2000; Salleh et al., 2010) and increases each student's individual degree of efficiency (Cockburn & Williams, 2000; Cliburn, 2003; Werner et al., 2004). Pair programming is also suggested to increase employability once students enter the workforce (Klopp et al., 2017).

Some research has been designed to determine the best ways to execute pair programming events. While the driver-navigator relationship may sound like a one-sided

conversation, pair programming events in which the driver speaks in equal amounts to that of the navigator are associated with better results (Rodriguez et al, 2017). This is especially true if the driver feels comfortable expressing uncertainty and the two resolve it together. Assignments that include an element of speed or place a premium on completing a task in the time allotted tend to result in more inequality between the participants wherein one partner becomes dominant and the other is marginalized (Lewis & Shaw, 2015). Several researchers have recommended ensuring that the students paired together have approximately the same amount of programming experience (Williams et al., 2006; Salleh, et al., 2010; Sobel et al., 2016). Others recommend that the instructor spend some time and thought teaching the students how to collaborate effectively (Webb, 2009; Kaendler et al., 204; Mercier et al., 2015). In the Computer Science Education research community, it has become generally accepted that pair programming is a useful tool for improving several dimensions of student experience and academic outcomes; however, this information does not tend to be disseminated among actual institutions in which CS is being taught (Winslow, 1996; Phillips, 2005; Clear, 2006).

Addressing Inclusion: Past Interventions

Some research that addresses the low representation of women and people of color in CS programs refers to stereotypes, as they draw from psychological studies of education. Students who do not fit established stereotypes of programmers are impacted academically (Lewis, Anderson, & Yasuhara, 2016) and self-assessments suggest they evaluate their own skills lower than equally-performing students who do (Lewis, Yasuhara, & Anderson, 2011). When students encounter stereotypes about computing, such as discourses relating to hackers or geeks, they will be more likely to leave (Peters, 2014). Questionnaire-based studies of college students in CS suggest that they feel stereotypes and socialized beliefs about who can be a computer scientist

affected their performance (Varma, 2010). Many of the authors investigating social challenges for CS students point to traditional socialization (Clausen, 1968; Macionis, 2013; Rodriguez & Lehman, 2017) and they agree that social construction is a useful tool for describing the specific difficulties that women and underrepresented minorities face (Brickhouse, 2001; Rodriguez & Lehman, 2017). Ambient belonging is also a relevant psychological concept to CS Pedagogy; a student's ability to thrive in an educational space can be affected by the belief that one's own cognitive abilities do not match those of the people already present in a space (Stout & Blaney, 2017). Students tend to lose confidence when they compare themselves against computing enthusiasts (Kinnunen et al., 2013). This is compounded by a documented tendency for teachers to direct their teaching towards students they presume are enthusiasts or have more experience programming (Ulriksen et al., 2015; Bhardwaj, 2017). Women in this field struggle with motivation and confidence (Nelson et al., 2013; Lishinki et al., 2016; Sax, 2017). Despite not mentioning gender bias as a problem, female students did report having had negative past experiences such as run-ins with aggressive or patronizing teachers, struggling to access school resources like lab stations due to more aggressive male students, projects that appeal more to masculine interests, or lack of female role models (Miliszewska et al., 2006). Students in CS tend not to express a "growth mindset," or the belief that knowledge is acquired through effort, and so when women feel they are exerting significant time and energy learning to program, they perceive it as a lack of natural fitness for the field and express low intellectual belonging (Stout & Blaney, 2017). This means that there is a discourse around programming that one either is or is not intrinsically, naturally, suited to the activity, which makes it easy to essentialize.

Some instructors feel they need to be more conscious of what kind of person they are asking students to be (Carlone & Johnson, 2007; Ong, et al., 2011; Rodriguez & Lehman, 2017).

Students have to choose between embracing and resisting very specific ranges of identities that can be a programmer (Eisenhart & Finkel, 1998; Brickhouse & Potter, 2001). CS identity has the strongest correlation with intent to continue in the CS major, and there is a significant gap between men and women's degree of programmer identity formation (Dempsey, et al., 2015). A sense of belonging can help students overcome a lack of confidence and increase persistence in the field (Goodenow, 1993; Veilleu et al., 2012). This can be done by managing the messages students receive concerning the acceptable range of identities for programmers, and classroom environment can also shape ideas about what programmers are (Schulte & Knobelsdorf, 2007; Zander et al., 2009; Wong, 2016; Hansen et al., 2017).

A popular strategy for counteracting harmful beliefs about the fitness of women for programming is to create programs that educate girls separately from boys (Hur et al., 2017). Often, this is put into practice in the form of programs that make the activity seem more feminine. Those who design learning software for girls are often influenced by gender-based stereotypes when doing so (Huff, 2002; Sele, 2012). The very existence of specialized learning programs that speak directly to girls actually reinforces a sense that technology is more naturally suited to boys and that girls are a special case (Cassell, 2002; Huff, 2002; Magee et al., 2011; Rode, 2011; Sele, 2012). This separation and modification of the learning process legitimizes perceived gender differences and reminds girls that to really belong they must reject femininity (Sele, 2012). The fact that there is a need to declare that girls can code, too, suggests that it is not a given. Girls' CS education tends to focus on building stereotypically feminine products such as social media sites and photo editing software, and promotional materials feature pink and floral designs, which reinforces the gender essentialism which originally contributed to the problem these programs are attempting to solve (Sele, 2012). While there are not as many programs that

focus on people of color as an underrepresented group in CS careers, it can be argued from these findings that focusing on essential differences between them and ratified members of computer culture would not yield positive results.

Several interventions have been shown to have a positive effect on academic outcomes, intent to continue study, and affect toward programming in women and students of color. When women have positive experienced in their first CS course, their intent to take another one is strengthened (Beyer, 2014). An increase in social support is considered an excellent direction for increasing the effectiveness of student socialization. Miliszewska et al. (2006) noted in a survey of Computer Science students that while female students do not explicitly recognize gender bias as a significant issue for their success, they report that they rely heavily on other female students for support and help. In a study of successful changes at Harvey Mudd College, Alvarado & Dodds (2010) found that several measures resulted in increased retention of female students. Moving the focus of the first course away from deep study of a programming language, they adjusted the goals of the class to include a broad introduction to computers that built from a set of skills developed in the first few weeks of class. They separated their sections based on prior experience so that those who had it were not held back and those who did not have it were introduced to important terms and concepts, and they included optional but incentivized lab sessions where students could work on projects with instructors. Female students were taken to a major conference so that they could see what kinds of research they could participate in as they continued in the program. Students were also given positions on research teams where they could develop their skills and gain confidence that they could already advance the interests of a professional project. These changes resulted in increased retention of female students, suggesting that support from peers and instructors, tangible successes and real-world application of their

knowledge contributed significantly to their desire to continue in the program. Ensuring that women in a small work group had at least one female peer with them increased their motivation, verbal participation, and intent to work in the engineering field (Dasgupta et al., 2015). A CS course that focused on generating only gender neutral and unbiased messages, incorporated an interdisciplinary approach to programming, and relied on everyday examples to demonstrate course concepts increased the likelihood of all students to complete the course, with a particularly strong effect for women (Svedin & Balter, 2016). For students of color, emphasizing the importance of representing their community in the field of programming increased their desire to pursue a career in CS (Lachney, 2017). This was found to be effective only in combination with explicit work constructing identities that involved programming as a part of their lives. A concern for students who do not identify with programmer identities is that their knowledge is delegitimized by the fact that it comes from an unratified member (Lagesen, 2007), meaning that any program that explicitly aims to reduce the power of those harmful cultural values should improve the experiences of women and people of color.

Contributions of This Study to the Literature

In reviewing the literature surrounding classroom practices, student outcomes, and the cognitive challenges of learning a programming language, several questions present themselves. As discussed in Chapter One, there are considerable issues at play nationally concerning identity formation and sense of belonging in students who do not arrive at the Introduction to Programming course with significant experience in programming already. These issues are particularly damaging to goals involving improving diversity in CS major enrollment. Comprehensive reviews of scholarship concerning CS students find that there is a need for

research that can provide a theoretical understanding of the process of computing identity development (Rodriguez & Lehman, 2017). This can be achieved through the application of sociolinguistic theories of language use and the interactional perspective it provides.

Studies investigating the success of pair programming in student outcomes, affect, and perseverance suggest that increasing collaboration improves some of the challenging conditions present in the programming classroom. These studies tend to capture only that its introduction brought on improvements, but do not have a granular, interactional approach to determining why this is the case. The success of pair programming in schools gives rise to an important question: why does having students talk more to one another lead to better outcomes? Why does more peer talk especially benefit women? These questions can be answered using the theories described in this chapter and the methods described in the following chapter.

The application of sociolinguistic theories and methods to Computer Science education is extremely rare. Many of the educators who research Computer Science education are not familiar with sociolinguistic methods that might illuminate the nature of learning to program, and sociolinguists who use ethnographic and interactional approaches are not often afforded the opportunities to conduct studies in which they can collect audio-recorded classroom events for the entire duration of a course devoted to a programming language they know. The corpus of data collected in the pursuit of the current study represents a rare opportunity to join the aims of the Computer Science Education community with the methods and analytical tools available in sociolinguistic research on learning. By applying language socialization theories and methods to the Computer Science classroom, this study can join national goals of encouraging diversity and inclusion, institutional goals of fostering improved student acquisition of terminology, and

interpersonal goals of increasing a sense of belonging for students who are beginning to learn how to program.

CHAPTER 3: METHODOLOGY - DATA COLLECTION AND ANALYSIS

This chapter describes locations of the Introduction to Programming course, its students, and its instructors. It outlines the activities that take place in each location, and which participants can be found there. Researcher role and data analysis methods are also discussed.

Setting

The University of California, Davis is a large public university in Northern California (UC Davis). It is attended by over 28,000 undergraduate students. UCD is ranked 11th nationally for public universities and offers more than one hundred majors. The Computer Science department is located in the College of Arts and Sciences. The introductory course enrolls approximately 300 students each quarter. The different types of activity that may constitute socialization to use relevant terminology are the following:

Table 1: ECS 10 Socialization Events Summary

Event	Setting	Socialization Activities	Participants
Lecture	Lecture Hall 50min 3x/week All students	Lecture Live-coding Students answering instructor questions Students asking questions publicly One-on-one questions (afterwards)	Professor Eiselt (Kurt) Students

Discussion Section	Classrooms 50min 3x/week Groups of ~20	Lecture Live-coding Students answering instructor questions Students asking questions publicly One-on-one questions (afterwards) Pair programming (Winter only)	TAs Students
Lab	Computer labs 3hrs 6x/week Optional for all	One-on-one questions	TAs Students
Homework	Home/campus	Individual work Peer work	Students (not observed)

These students attend three hours of lecture and one hour of discussion section each week. Professors and teaching assistants offer office hours in which students can ask questions about the course materials and show their programs to receive comments and corrections. The professor who instructed the course told students both quarters that his office hours are for general advice about majoring in Computer Science, while the TA office hours (referred to as "lab") are for reviewing course material and correcting programs. There is also a Computer Science Club that offers tutoring and review sessions in Kemper Hall, where the instructors' offices are.

The Lecture Hall

During the two quarters, students attended lectures in a large hall three times per week. In Fall quarter, 357 students were enrolled in the course; in Winter quarter the number was 257. The lecture hall was therefore smaller in Winter Quarter. For both quarters, students arrived between 9:50 and 10:00 in the morning on Mondays, Wednesdays, and Fridays, found a seat, and

directed their attention to the front of the hall, where Professor Eiselt (Kurt) would stand at a podium. A large projection screen would lower from the ceiling in the center of the front wall. Fall quarter saw the students in a large cement auditorium-style room with rows of seats on tiered steps. Two sets of steps formed aisles between the seats. In Winter, lecture took place in a smaller room, which was carpeted and considerably darker than the auditorium. Kurt stood much closer to the front row of students in this room.

Sometime between 9:50 and 10:00 each morning, Kurt would arrive with a colorful backpack, make his way to the front of the room, and pull out a MacBook and a VGA-Mini DisplayPort adaptor. He connected this to the cords on the podium and craned his neck to see whether the image on his screen was being correctly displayed on the projection. Often, there would be issues concerning color, size, or consistency of transmission. Kurt typically expressed his frustration, causing a ripple of laughter among the students present.

At about 10:00, he began lecture, most often by exclaiming, “Good morning!” Then he would give a summary of the upcoming topics for the day. The lecture itself consisted of varying proportions of spoken information paired with a slide containing the same information, more informal example description, live coding, and time during which students were to try coding a solution themselves. Kurt’s laptop displayed a PowerPoint presentation, which he sometimes minimized in order to pull up the Python development environment for live coding. When lecture concluded, typically at precisely 10:50, students would file out of the auditorium except those who wished to ask a question. Those students formed a line at Kurt’s podium. If the questions continued for more than five or six minutes, Kurt would ask the students to come to his “office,” by which he meant to stand with him just outside the lecture hall. This would continue

for at most five or six more minutes, at which time the students would all have dispersed and Kurt returned to the Computer Science building.

The Discussion Section

Each quarter, students were assigned to attend one discussion section per week. There were six sections each quarter. Students were not required to attend, so attendance varied throughout the quarter. Discussion sections took place in various classrooms throughout campus. Students primarily attended the section to which they had been assigned, but some attended others due to schedule preferences or if they preferred one teaching assistant (TA) over another. For Fall quarter, the TAs were Anshika, John, Chris, Gwen, and Tom. For Winter quarter, the same TAs taught discussion sections, with the exception of Gwen, who was on a fellowship and did not teach. The focal TAs were Anshika, John, and Chris because they participated fully in every aspect of the experiment.

Table 2: ECS 10 Discussion Sections Schedule

	Section	Day/Time	TA
Fall 2016	A01	Wed. 3:10 - 4:00 PM	Tom
	A02	Wed. 9:00 - 9:50 AM	Tom
	A03	Fri. 4:10 - 5:00 PM	John
	A04	Thurs. 8:00 - 8:50 AM	Dana

	A05	Fri. 9:00 - 9:50 AM	Anshika
	A06	Tues. 8:00 - 8:50 AM	Chris
Winter 2017	A01	Mon. 1:10 - 2:00 PM	John
	A02	Thurs. 10:00 - 10:50 AM	Tom
	A03	Fri. 3:10 - 4:00 PM	Anshika
	A04	Wed. 4:10 - 5:00 PM	Chris
	A05	Mon. 12:10 - 1:00 PM	John
	A06	Thurs. 6:10 - 7:00 PM	Tom

Discussion sections took place at various times of day and days of the week. The classrooms typically had a capacity of approximately 40 students. TAs stood at the front of the room with a tabletop lectern and a projection screen that descended from the ceiling. Students sat in chairs with built-in desks. As their baseline activity, some TAs used PowerPoint slides describing course concepts. Some preferred to display the students' homework assignments and the Python development environment. Discussion section consisted mainly of live coding on the laptop or writing pseudocode on the whiteboard or blackboard. TAs asked students to report on course concepts from the preceding lectures, then illustrated those concepts using live coding or pseudocode. Students asked for guidance on their homework or clarification of course concepts, which TAs explained, often with the use of live-coded examples. Before a midterm, TAs read over the sample midterm and answered the questions for the class. After a midterm, they did the

same with the actual test. Midterms were passed back in this setting as well. As in the lecture, students lined up at the lectern to ask questions after section was over.

The Lab

In addition to lecture and discussion section, TAs held office hours in computer labs on campus for three hours at a time. These were rooms containing long tables with rows of computers, which students could use to work on their homework assignments or study for exams. Most students worked on their own laptops in this space, although some used the large desktop computers provided. The TAs would sit at the front of the room until someone needed help, at which point they would raise their hand and the TA would approach and ask what they needed. At the end of each quarter, attendance at labs surged and the TAs instituted a waitlist: students wrote their names on the board and the TA called out the names as they finished helping the previous student. In the last week, TAs would sometimes not make it through the whole list. Those students whose names did not get called would have to find another way to get help - typically this meant corresponding over email.

Table 3: ECS 10 Lab Schedule

Quarter	Day/Time	TA
Fall 2016	Tues. 4:00 PM - 7:00 PM	Chris
	Wed. 11:00 AM - 2:00 PM	Chris
	Thurs. 11:00 AM - 2:00 PM	Anshika
	Fri. 1:00 PM - 4:00 PM	John

Winter 2017	Wed. 11:00 AM - 2:00 PM	Anshika
	Wed. 4:00 PM - 7:00 PM	Chris
	Thurs. 11:00 AM - 2:00 PM	Anshika
	Thurs. 4:00 PM - 4:00 PM	John

Participants

The population of this study was undergraduates enrolled in ECS 10 (Introduction to Computer Science) in Fall 2016 and Winter 2017, their TAs and their professor. In Fall 2015, I met with Nina Amenta, the chair of the Computer Science department at UC Davis, about studying the students and instructors of the course the following year. She expressed interest in the language socialization process in the course, and she asked that I observe and record the students in order to give advice concerning increasing student acquisition of class-related terms. Later that year, she invited me to a meeting with Kurt Eiselt, who would be the instructor of record for the course during Fall and Winter of the following year. He was pleased to invite me to observe his class, provided the TAs consented to observation. He was interested in student acquisition of Computer Science terminology, but he also wished to know what he could do to ensure that women and people of color were encouraged to continue in the major after completing his course. In the week before instruction began in Fall 2016, I met with Kurt and the TAs to explain my study to them and present the literature I had collected suggesting that pair programming could be a viable method for increasing language socialization and desire to continue taking Computer Science classes. They agreed to participate in the study as well.

On the first day of instruction in both quarters, I introduced myself to the students in lecture and handed out copies of an IRB-approved informed consent document. Of the 614 total undergraduates enrolled during those two quarters, 543 consented to be observed and recorded. Instructors and students read and signed IRB-approved informed consent documents, and students also signed a letter to the UC Davis registrar allowing me access to their grades and majors.

Instructors

The professor for ECS 10 was Kurt Eiselt, who had moved to Davis from Canada the previous year. He had completed his undergraduate degree in California. He taught the course both quarters. Kurt was a middle-aged white man who had taught Computer Science for several decades. In conversations with me, he often made comments that indicated extensive experience with students in large lecture-based classes such as ECS 10. Kurt interacted with the students during lecture, after lecture, in his office hours, and over email. He originated the lecture contents and wrote the exams and assignments. He selected the textbook, *Think Python, 2nd ed.* By Allen B Downey.

The TAs in ECS 10 were John, Anshika, Chris, Tom, and Gwen[1]. Because Gwen did not work with students both quarters, her section was not included in the qualitative data analysis. Because Tom did not participate fully in the pair programming activities, his sections were also excluded. John, Anshika, and Chris were all Masters students at UC Davis at the time of data collection. They had never taught a programming course before, and they all had varying degrees

[1] The names of all TAs and students are pseudonyms, but Professors Eiselt and Amenta have elected to be identified.

of experience with Python. All three TAs taught one to two discussions and supervised one to two labs each week. They also corresponded with students over email.

John was a 23-year-old Chinese man who had lived in California and Washington. In their questionnaire, students largely described John as friendly and knowledgeable. He often stayed after second for up to an hour to answer the questions of his students, which made them feel cared for. One student wrote, "He tells us how he is going to grade our homework. I love that Leo really cares about us students, and i love that he is willing to stay after class to help them out. He really made the class enjoyable for me." They sometimes wished he would do his live coding in the Python development environment, rather than writing it out on the blackboard. For example, a student wrote, "It would be more helpful if he codes on the projector instead of writing on the board." Many students felt he spoke too quickly and too quietly, and a few mentioned his English. John's English was highly proficient, although his speech was characterized by L1 interference, mainly in the form of deleted articles and lack of subject-verb agreement. A student wrote "His voice was a little quiet (maybe I should have sat in front) and his English could be slightly hard to understand at times." For example, on December 12, 2016, John displayed these linguistic variables while demonstrating the use of dictionaries:

> John: So, uh, **dictionary two is empty one**, and **dictionary one is have** like two things in it but what we want is, if **the dictionary two don't have** that element, we want it to set a, uh, default, which is zero, but if there i- there is a A, uh, if there, uh, dictionary two is like A (.) one.
>
> Audio-Recorded in John's Section 12 December, 2016

While John's non-native speaker status could have interfered with students' comprehension (Rubin, 1992; Lindemann, 2000; 2002), his students appeared to perform approximately as well as those in Chris's class. The same goes for Anshika, whose English was Indian-accented.

Anshika was a 24-year-old Indian woman who spoke extremely fluent Indian English. In their questionnaires, students described her as nice but not engaging or confident. One student wrote, "I don't like that she's not very clear. She also doesn't seem very confident or sure of herself. So there's not much structure to her discussions." She spoke to me about her concerns about getting students to speak during section, and she had received advice from a professor to pose questions to the class and then wait for an answer, even if it took a while. Students did not appreciate this technique; one wrote: "I didn't like how she would try to force participation when it was clear that a more lecture style format would have been better." Students disagreed on her ability to answer questions. Some said she did not fully answer the questions they asked, while some felt she was very patient and engaging one-on-one. One of the students' favorite things about her was the way she would demonstrate several ways to write a program that does the same thing, and then she would show how the program would execute if she made several different mistakes.

Chris was a 23-year-old white American man raised in northern California. The vast majority of students mentioned his friendliness and clarity of explanation. Wrote a student, "I enjoy the slides on the projector. I also enjoy him leaving the last 15 minutes of discussion up for dialogue and questions among the class to get one-on-one help." Students enjoyed his conversational, casual style, especially the way he would narrate his own problem solving process as he introduced concepts. A student wrote about this, "He writes out problems while explaining his thought process for how to solve them which helps me to metacognitively think about my own thought process and improve my efficiency for problem solving."

Students

In Fall 2016, there were 29,111 undergraduate students enrolled at UC Davis. Of these students, 94% were younger than 25 years old. Using the UC Davis Student Profile, I will compare the general undergraduate racial and gender proportions to that of the participants in my survey of ECS 10 students. My questionnaire did not treat international students as a separate racial category. Also, the university report includes middle eastern students in the category "white," while I created a separate category to reflect their status as a separate social concept.

Table 4: Demographic Information: UC Davis

	UC Davis		ECS 10	
Men	11,988	41%	144	56%
Women	17,558	59%	113	44%
African American / Black	1,049	4%	2	<1%
American Indian / Alaskan Native	241	1%	0	0%
Asian / Pacific Islander	10,337	35%	165	64%
Hispanic	6,206	21%	30	12%
White	7,781	26%	49	19%
Unknown	497	2%	12	5%
International	3,445	12%	NA	NA

The gender and racial makeup of ECS 10 does not appear to be representative of the general undergraduate population of UC Davis. This conclusion was corroborated by eye in my field observations. Where women outnumber men in the general population, ECS 10 contains a disproportionately high number of men. Black and Latinx students were dramatically

underrepresented in the course (in fact, there were no Black students at all enrolled during Winter quarter), while Asian and Pacific Islanders were overrepresented. This disparity in enrollment is reflected in surveys of computer science programs in the United States in general (CS Education Statistics).

The students who enrolled in ECS 10 were all 24 years of age or younger. They were enrolled in a variety of majors. I observed them in lecture, discussion sections, lab, and on the class website chat room. Students who participated often in lecture, section, and lab became focal students, primarily because of the volume of data they provided.

Researcher Role

My data collection methodology was participant observation. I attended one to two lectures and two to three sections each week. Depending on the time of quarter, I attended one to three lab sessions each week. In lecture, I laid my recorder at the front of the room and sat in a student seat with my laptop open to take notes in SublimeText. Approximately 40-60% of students had their laptops open during lecture as well, so I was a relatively unremarkable presence in that room. In discussion section, nearly all students had their laptops open, so I was in the same position.

During lab, I stood near the front of the room and watched for students who needed help. Early on, students began asking me for help as well. At first I hesitated to assist them, but the TAs were grateful for my help because it allowed them to spend longer speaking with each student. Initially, I had erroneously believed that I had significantly less knowledge of Python than the TAs. I learned over time that while some of them had used it for much longer than I had, John had used the same online course I had, and Chris had not spent much time with it at all. In

addition, the TAs were not required to attend lecture, so I was the only one who knew exactly how Kurt had presented the course material. This allowed me to ensure that what I said to the students was extremely similar to the words Kurt had used. In fact, I made it a point to try never to say anything I had not heard Kurt or a TA say, so as to avoid accidentally giving the students too much information. John, who had the most-attended lab hours, seemed to become somewhat dependent on my help as his student volume increased. He would ask me to come help him on days he knew would be especially high-volume. I did my best to oblige, out of equal parts enjoyment, desire to collect more data, and desire to help a friend. The other TAs did not request my presence, but were happy to have me there, with the exception of one. As I had become aware, researchers can sometimes be positioned as spies or troublemakers (Duff, 2017), and the reticence with which Tom allowed my presence in the classroom suggested to me that this was his position. This coupled with his vocal resistance (in and out of the presence of students) to the idea of instituting a pair programming activity in sections, as well as his tendency to find reasons to skip that particular activity wherever possible, caused me to remove his sections from the data analyzed in this dissertation.

Because I was interested in the students' learning experiences as well as helping them with their assignments, students in lab tended to be more casually conversational with me than with their TAs, leading to conversations about stress, nerves, and plans for their future study. Cristina, for example, attended so many lab sessions that she became extremely accustomed to my presence and spoke frankly and casually with me about her experiences in the course.

The TAs also became extremely familiar with me. Because I was with them for so much of their instruction time, I was the same age as they were, and I had been a TA for several courses myself, we had quite a few topics of conversation about which we felt comfortable

speaking during down time. John and Chris especially enjoyed chatting with me before or after sections; John about his job search (he was hoping to move to the Bay Area, where I live, to work at a large technology company) and Chris about his girlfriend and their dog. Anshika was quite focused and quiet when I observed her, but she was interested in hearing about my progress planning my wedding (I had gotten engaged three months into the six-month data collection process). My relationship to these TAs evolved into something very similar to the relationship I have to other graduate students in my department.

Data Collection

The design of this study consists of two parts. The first part is an ethnographic observational study in which I established a baseline observation of attitudes, discourses, language socialization, along with measurements of academic performance and persistence in CS. Part two is an experimental study that manipulates the degree to which students are asked to collaborate during the quarter, and changes in student attitudes, language socialization, academic performance, and plans to continue on the CS track are investigated in conjunction with more stable relevant personal information such as previous programming experience, gender, race, nationality, and age.

Observation Types, Locations, and Times

In both phases, student identities and discourses were analyzed using questionnaires in which I measured demographic information such as age, race, gender, and major. I also used the questionnaires to determine how much experience each student had had with computer science and how likely they felt they would be to continue taking CS courses. Students were given a

chance to describe what they liked and disliked about their instructors' methods, and they answered Likert-scale questions rating their familiarity with the material, experience programming, confidence in their future success, sense of belonging, and degree of positive face when interacting with instructors.

I also observed the students, instructor, and TAs of ECS 010. During lecture and discussion sections, I sat in a student seat near the front or back of each classroom in order to observe. I used either my smartphone (running Voice Notes) or a USB voice recorder to take an audio recording of the class. On days I was not able to attend lecture, Kurt recorded himself with a USB voice recorder I supplied. During lab, I stood near the TA-student group in order to capture their audio as they discussed the student's program. I took field notes in a text file on my laptop. Ethnographic field notes involve quick, general note-taking in the moment, sometimes supplemented by audio recordings. Directly after the lesson or lab time is over, I read over the notes, adding detail, context, and some immediate reflections in the tradition of ethnographic field notes (Schatzman & Strauss, 1973).

The experimental phase of the study involved introducing a pair exercise into the students' discussion section. Previous studies have identified the addition of opportunities to collaborate as a significant factor in student success (McDowell et al., 2002; Hancock, 2004; Barker, et al., 2009; Alvarado & Dodds, 2010). I worked with Professor Eiselt, the instructor of the course, to change the syllabus in Winter Quarter such that students were given time in their discussion sections to do pair programming. Students were given the opportunity to choose their own partners, as Professor Eiselt believed this would increase their enthusiasm for the activity. They worked together, alternating between themselves the role of *driver* and *navigator* (Cockburn & Williams, 2000). The driver typed answers on one of the laptops, but they were not

allowed to type without explicit instruction from the navigator. This activity required students to transmit what they already knew to one another and develop new skills between themselves rather than relying on the instructor to show them. It is well-suited to language socialization research because of its similarity to interactional perspectives on task completion (Ochs, 1991). In order to ensure that the TAs knew how to moderate this collaborative experience, I explained the procedure in discussion sections so that the students and TAs were all comfortable with its execution. I used the guidelines established by Laurie Williams, the originator of this practice (Williams, 2000). I compared the performance of the students enrolled in Fall quarter to those in Winter Quarter to measure differences in performances that could be attributed to the new requirement I introduced in winter. I also compared student-teacher dyads to similar speech acts in peer dyads in order to describe the differences between instructor-student socialization and peer-to-peer socialization.

Data Analysis

In order to describe processes of language socialization into course-related terminology and the social category 'programmer,' I used discourse analysis, pragmatic analysis, and statistical analysis. These three analytical methods allowed me to create a holistic description of the actions and events that shaped students' experience of the course and sense of belonging in that context.

Ethnographic Analysis

The role of ethnography in education is the same as the role of ethnography in any field: "to document the existence of alternative realities and to describe these realities in their own

terms" (Spradley, p. 14). Ethnography is a way for education researchers to enter the world of the classroom and develop theories about learning and teaching that originate from the creators of that world. Ethnographic methods are a common way to generate data for analysis using Language Socialization as a theoretical framework (Hymes, 1964; Gumperz & Hymes, 1972; Heath, 1983; Duff, 2011; Ochs & Schieffelin, 2017, e.g.). The application of detailed discourse analyses to naturalistically observed and recorded language excerpts create a strong relationship between this methodology and Language Socialization Theory (Garret & Baquedano-Lopez, 2002; Kulick & Schieffelin, 2004; Duff, 2008a).

By participating in the ECS 10 class for the duration of two quarters, I was able to develop a depth of understanding unlike that provided by questionnaires or assessments alone. I observed, took notes on, and transcribed the events of the course, then I coded those notes using thematic coding. I also annotated each class term as it was referenced so I could collect those terms together to track its life cycle within each quarter.

As I coded the data, I created codes related to practices and beliefs. The socialization practices that emerged were lecturing, live coding, walking through functions, asking for guidance, and testing/debugging. An excerpt was coded as "lecturing" when an instructor spoke in a monologue about a term or process. "Live coding" was the code for instances in which an instructor typed out code on their laptop, displaying it on the projector. The code "walking through functions" describes moments in which an instructor displayed a complete function, imagined a value to assign to the input variables, and then demonstrated how those variables changed during the execution of the function. These were all practices that were primarily the domain of the instructors. The next code, however, came from students: "asking for guidance." This might take the form of asking a factual question, asking for advice, or showing deficient

code in the hopes of instructions. Finally, "testing & debugging" was practiced by all participants: this code was used when a participant ran their code or re-read it in the hopes of discovering any errors or opportunities for improvement.

As I was interested in discourses describing the cultural concept of "programmer," I also coded for socially ascribed identities, when they appeared. These were teacher, student, programmer, nerd, masculine, feminine, old, young, local, and foreign. Whenever a participant positioned themselves in terms of one of these identities, the relevant code was assigned. An example of a transcription that would be coded as "old" is a moment in January 2017, when Kurt made a point about nerves:

> Kurt: Who's nervous? Who's worried? Come on, more of you. You know that's true. Lot of you are just lyin'.
> Some: ((laugh))
> Kurt: When I was in your position,
> Some: ((laugh))
> Kurt: before the dawn of history,
> Some: ((laugh))
> Kurt: I was nervous. I- I'd never done this stuff before. Right? But it was okay! It was fun! Don't be nervous.
>
> Audio-recorded in lecture, 1/11/17

To put the students at ease, he made a joke at his own expense, calling himself old. I coded instances of both reflexive and interactive positioning using these themes.

Discourse Analysis

I performed discourse analysis on the field notes, transcribed audio, and free-answer questionnaire items in order to describe the character of the field students believed they were entering, as well as the ways students resisted or reproduced the discourses around race, gender, and programming. This analysis followed the framework described by Teun van Dijk (1997). Discourse analysis often relies on transcribed recordings of conversations so that the researcher

can be more sure of a faithful representation of what was said and how. With a small team of student research assistants, I transcribed the audio recordings from lectures, discussion sections, labs, and brief conversations with the instructors. Transcribing conversation requires the transcriber to make decisions about which behaviors should be represented (Mishler, 1991). I developed transcription conventions similar to those of James Paul Gee (2014), who focuses on linguistic strategies and events such as repairs, false starts, overlapping speech, and changes in pitch. These conventions can be found in Appendix A.

Discourse Analysis involves taking these linguistic strategies and events with their contexts and effects, using them to construct the reality of the participants according to what is considered normal and right within that community and what is outside the bounds of the normal for them. In the process of analyzing the discourses that are supported, resisted, and transformed during the course, the researcher must allow participants to be the genesis of conclusions concerning their identities and ideologies (Bucholtz & Hall, 2004). This means, for example, identifying moments of differentiation to define what identities are marked or unmarked in the given context (Urciuoli, 1996; Spitulnik, 1998; Bucholtz, 2001; Bucholtz & Hall, 2004).

This method also contributes to an understanding of the nature of the language socialization that took place in the ECS 10 course. The transcribed audio recordings allowed me to use my observations to draw conclusions about the degree to which students interacted with each other and perform close analysis of language change over time. I selected two of the most discussed concepts in the course and tracked the terminology as it was introduced, used by instructors, and then taken up by certain students. In addition to interpreting the *product* of language socialization - changes in language- I "examine [the class's] mental process from the point of view of the *activity* taking place and ask what kind of social actions and cognitive skills

constitute that interaction" (Ochs, 1991, pp. 144-5). This task-based approach directed the analysis of language socialization so as to align with the higher education context in which it took place. I searched for patterns in the students' speech to indicate whether they were taking up and integrating the terms as they were presented in class and lab.

In addition to the socialization of students into proper use of terminology, I used discourse analysis to analyze the construction of the programmer identity. As discourses are enacted, they rely on meaning, style, rhetoric, and schemata (van Dijk, 1997). Discourse analysis allowed me to take a microscope to the ways students and instructors constructed identities through individual practices that intersected on multiple social dimensions (Eckert & McConnell-Ginet, 1992). As language socialization can be seen as a process whereby novices negotiate social identities (Schieffelin & Ochs, 1986), this is done partly through appropriation of and resistance to discourses (Mendoza-Denton, 2008; Duff, 2010). Along with course concepts, students learned the sociocultural concept 'programmer,' assessing whether it fits their own self-concept. Discourses defining who naturally fits the social category of 'programmer' can provide insights into the sense students have that they will belong in the Computer Science major.

Conversation Analysis and Pragmatics

Once I located important moments of language socialization in the transcribed interactions, I used Pragmatic Analysis to come to conclusions about students' and instructors use of Face-Threatening Acts with or without redressive strategies. Levinson (1983) considers pragmatics to be an important part of Conversation Analysis, a viewpoint that is not necessarily universal (Drew, 2017) but which, however, provided useful tools for discussing the

management of identities, discourses, and face in the context of conversations and lectures. Analyzing apparently minute choices in wording, syntax, pitch, volume, gesture, and more, allows the researcher to identify moments in which routine behaviors are established and maintained (Schegloff & Sacks, 1973; Sacks, Schegloff, & Jefferson, 1974; Schegloff, 1982, 1986), including the management of identities. Conversational analysis allows the researcher to identify the forms involved in co-construction at the most microscopic level (Goodwin, 1979;1994; Goodwin & Heritage, 1990; Goodwin & Goodwin, 1987). Applying this method to the co-construction of knowledge can show how participants recruit linguistic, paralinguistic, and non-linguistic actions to play out, reaffirm, challenge, maintain, and modify social identities (Jacoby & Gonzales, 1991; Ochs, 1993; Jacoby & Ochs, 1995).

Conversation analysis makes use of pragmatics as one of the tools for locating the schema participants draw upon to choose what they say (Drew, 2017). I used Speech Act Theory, an analytical framework within Pragmatics, to create categories describing common linguistic events in the ECS 10 class. Brown and Levinson's Facework, introduced in chapter two, is the theoretical framework on which I base my analysis of the speech acts present in the transcribed recorded data I collected. Face threat has been used as an important part of Language Socialization studies (Tracy, 1997; Duff, 2009; Duff, 2010) as participation in activities requiring oral discourses involve serious conflict, tensions, loss of face (Tracy, 1997). Participation in class constitutes an important classroom behavior, but it is also a threatening one that requires face threat redressive strategies (Frisby et al., 2014). The speech acts I selected for further analysis are Asking For Clarification and Correcting Errors. The act of Asking for Clarification takes place in the classroom setting and individually, which was an area of interest in terms of what FTA redressive strategy was most preferred in those two contexts. Asking an

instructor for clarification presents a negative face threat (asking the hearer to spend time further explaining the concept) and a positive face threat (challenging the quality of the hearer's previous explanation). Because students were not responsible for explaining terms or concepts to one another, asking a peer for clarification only presents a negative face threat. This change in threat level is also of interest in terms of the effect on student acquisition of terminology. The act of Correcting Errors presents a negative face threat (asking the hearer to change their program) and a positive face threat (challenging the hearer's knowledge or skill). This is true regardless of whether the speaker is an instructor or a student.

For the purposes of this study, Brown and Levinson's (1978) use of positive and negative face are the primary lens through which I viewed student-instructor and student-student interaction. Arundale (2010) used this framework in conjunction with a conversational analysis methodology in order to show how face wants can interact with relational connectedness. Brown and Levinson's strategies for redressing FTAs fits comfortably with this more recent interactional identities research. In order to perform speech acts, speakers are often required to threaten hearers' face. Face-threatening acts (FTAs) that could potentially damage a hearer's positive face are criticisms, raising inappropriate topics, belittling, boasting, interrupting, non-sequiturs, mis-addressing, among others. Speakers might endanger their own positive face by apologizing, accepting a compliment, confessing, and losing physical or emotional control over themselves. FTAs that could potentially damage a hearer's negative face are orders, requests, advice, threats, warnings, compliments, offers, promises, and other acts that impose S's will onto H's time or freedom. Speakers can put their own negative face in danger by thanking someone, accepting an apology or an offer, committing themselves to an unattractive activity, or other acts that sacrifice their time or freedom.

An FTA can be redressed with strategies that give face to the hearer as they are committed. Brown and Levinson list five strategies to reduce the effects of a FTA, in ascending order of effectiveness: (a) commit the FTA without any minimizing strategies, (b) commit the FTA on-record, accompanied by positive politeness, (c) commit the FTA on-record, accompanied by negative politeness, (d) commit the FTA off-record, or (e) do not commit the FTA at all. Should S choose too mild a redressive strategy, H could take offense at S's apparent disregard for their value. Should S choose too strong a redressive strategy, H could believe the threat to their face is much greater than S intended. Brown and Levinson derived a computation of how weighty an FTA might be based on several variables (W_x). The equation illustrating this computation is as follows:

$$W_x = D(S,H) + P(H,S) + R_x$$

D(S,H) represents the social distance between the speaker and the hearer. This is a symmetric measure, and according to the authors it represents the frequency of interaction and the quality thereof. The definition of 'distance' is not always consistent from author to author (Spencer-Oatey, 1996). Some measure it by length or frequency of contact (Slugoski & Turnbull, 1988), some by how well the participants know each other (Spencer-Oatey, 1996), and some by how much they like one another (Baxter, 1984).

P(H,S) represents the relative power of the hearer over the speaker. Brown & Levinson define this power as the degree to which H can impose his own plans and self-evaluation at the expense of S, as well as the authorized or unauthorized material control H has over S. Similarly, the definition of 'power' can be inconsistent between authors. For some, power of control is the primary measure for power (Brown & Gilman, 1972; Brown & Levinson, 1978). Leichty & Applegate (1991) only consider power difference to be a factor if the power of control is

legitimate, or sanctioned. For others, social status or rank is more important than actual power to influence the actions of another person (Holtgraves, 1986).

Finally, R_x refers to the absolute rank in the participants' culture of possible impositions. This is measured in terms of how much X impinges on H's negative or positive face wants. It can be affected by how much time, money, or effort H would have to expend to perform X, whether H has a legal, moral, or employment-related obligation to do X, whether H would enjoy X, or whether there are any compelling reasons for why H could not or should not perform X. The three considerations – social distance, relative power, and absolute rank – are added together in the participants' minds as a method for evaluating how H will respond to the FTA. The speaker then selects one or more strategies in order to reduce the perceived value of one or several of these variables.

Items as minute as individual discourse markers can provide important information concerning the decisions that interlocutors must make as they negotiate meaning with each other (Lakoff, 1973; Schiffrin, 1983; Ochs, 1993; Beach, 1993; Fraser, 1996; Fuller, 2003; Bolden, 2009; Othman, 2010; Fraser, 2011). For example, tag questions and particles have been observed in conversation as requests for confirmation, whose purpose is to ensure in the midst of an explanation that the hearer is understanding and agreeing with the speaker (Lakoff, 1973; Dubois & Crouch, 1975, e.g.). Affirmative and negative particles are also frequently observed as marking the hearer's stances regarding the speaker's utterances (Ochs, 1993), and markers observed for epistemic stances of certainty or uncertainty (Levinson, 1983; Givon, 1989) include modals, rising intonation, and interrogative construction (Lakoff, 1972). People also mark for affective stance (Ochs & Schieffelin, 1989; Besnier, 1990; Irvine, 1990) using features such as vowel lengthening, modulation of volume and pace, morphological markings (e.g., using a plural

marking for a single referent), or code switching (Stankiewicz, 1964; Duranti, 1984; Ochs & Schieffelin, 1989). Self-initiated self-repairs arise when a speaker discovers they have spoken in error and interrupt themselves to produce the more proper utterance (Fromkin, 1971; Hutchby & Woffitt, 2008). A non-disruptive amount of repair in speech is a normal way of detecting and remedying problematic speech (Sacks, Schegloff, & Jefferson, 1974) but a disruptive amount could suggest hyper-concern for speaking properly. Pauses and filler markers indicate a similar desire as they are also a type of disfluency (Sacks, Schegloff, & Jefferson, 1974). These filler markers could also be described as prosodic hedges, which indicate tentativeness when threatening the negative face of an interlocutor (Brown & Levinson, 1978). Backchanneling is a way of indicating engagement to a speaker without interrupting them: for example, saying "mhm" while S tells H a story (Prepin, Ochs, & Pelachaud, 2013). Backchanneling indicates an affective stance that validates the speaker, conveying attention and agreement. These methods for indicating uncertainty, among other stances, can provide insight into the face considerations speakers are aware that they need to account for.

Statistical Analysis

In order to ensure that increased social engagement truly correlated with improved performance, I compared the students' grades to the other measurable values gleaned from the questionnaire. The dependent variables are between-subject factors: these are the students' final grades and whether they enrolled in a future ECS course. I compared these measures between quarters in order to determine whether the interventions had a significant effect on those variables. In measuring factors relating to the students' final grades, an ANOVA coupled with Tukey's range test was most appropriate. A successful treatment would result in decreased

significance of race and gender as a predictive factor in students' final grades. A successful treatment would also result in an increase in students who planned to enroll in another ECS course. I acquired this information by submitting a request to the UC Davis registrar during the quarter following observation. For this, I employed one-way ANOVAs to determine whether there was a significant relationship between quarter of enrollment and plans to continue in computer science.

Conclusion

When students enter the introductory course of a particular major, they are attempting to determine how successful they will be if they continue along that path. This decision-making process involves several considerations, among which are the student's own aptitude for the subject, the student's enjoyment of the subject, and the student's sense that there will be a fulfilling future for them in whatever career will result from this field of study. Students get a sense of these factors from their own academic performance, their environment, their learning materials, their peers, and their instructors. By applying language socialization, discourse and conversational analysis, and statistical measures, many of the considerations enumerated above can be measured and discussed.

In Chapter 4, I employ Language Socialization Theory as it pertains to students' negotiation of the meaning and use of new class-related terminology. In order to do this, I apply conversation analysis methods to uncover the mechanisms by which expert-like speech is modeled and taken up by the participants. This method is applied to the modeling and uptake of the terms "function" and "index" as examples of language socialization to new terms at two different points in the course. In Chapter 5, I discuss the use of politeness between instructors

and students as they accomplish the culturally relevant acts of asking for clarification and correcting another's error. The negotiation of face wants in these helps shed light on the effects of interpersonal and social factors on the ways in which this socialization is carried out. In Chapter 6, I recruit discourse analysis methods to define the larger discourses about who can be a programmer, and I present evidence as to the degrees to which the instructors and students reproduced, resisted, or transformed these discourses. I also report on statistical analyses of student grades and retention in the CS department.

CHAPTER 4: LANGUAGE SOCIALIZATION TO NEW TERMS

In this chapter, I describe the modeling, feedback, and uptake of student academic language using the terms "function" and "index" as examples. To describe the ways in which students incorporated new words into their linguistic competence in all of their morphological and syntactical variations, I employ *metalinguistic coercion* (Wilson, 2012; Duke, 2017) as an analytical tool. Metalinguistic coercion is the incorporation of object-language, or the language a speaker is attempting to master (Gombert, 1992) into a meta-language, or talk referring to the use and meaning of language (Duke, 2017). Students learning to program must learn new terms, which can be categorized into several types: new words (e.g., "concatenate"); known words used in new ways (e.g., "function" to mean a collection of code rather than a purpose), and known words with new derivations (e.g., "def" as short for "define"). Learning a completely new word would necessarily involve developing knowledge of the semantic, syntactic, and morphological elements that can be used appropriately, but learning to use the other two types of terminology may involve coercion. The professor was careful to use repetition and several modes of demonstration to express the boundaries of the meaning space for each word, and the TAs did this as well, albeit to a lesser extent. However, instructors were often unaware of potentially confusing or inappropriate uses of the terms that students were meant to take up and use at the level of a full member, and they missed several opportunities to model the proper use of terms or require that students use them. Students often exhibited avoidant behaviors where a target term was relevant, which was supported by instructors.

Functions

The term "function" was introduced early in the course and used at an extremely high frequency throughout the quarter. Students were asked to identify functions in the first or second week, and needed to write a function by the third week. From the third week on, all homework assignments required them to write one or more functions. A function is a block of code that executes a particular goal. There are pre-existing functions that can be used any time, and people can write new functions that can be used as long as they can direct the computer to where that function is stored. A function has a header, in which the name of the function and its expected inputs are written, and a body, in which the instructions for the function are written. To use the function, one must "call" or "invoke" it, meaning that the name of the function and its input values or variables are written out, after which the program will follow the instructions using those input values. The results of those instructions can be stored in a variable that the program or programmer can access by referencing that variable. The default result is called "None," which means nothing is stored, but the programmer can write a "return statement" at the end of a function to instruct the function to use some value derived from the instructions as its result. For the purposes of this course, functions can do several things, but they can only return one result.

Modeling "function"

In the fall quarter, the term "function" was not officially introduced with its definition before instructors began modeling its use. Students might have read its definition before class if they had read the textbook (most students reported not reading the textbook unless they were stuck on a homework problem, however, so this is relatively unlikely). In the fall, Anshika's and Chris's students heard the term in discussion sections; the rest heard it first in lecture.

Chris, who was describing the interactive development environment, a space in which students could type out code and see it executed live:

> Chris: It's more useful for just testing small things, um, if you want to do, like, a couple of operations, like let's say, um, I want to get some input from the user, right? So we're just going to use the **input function** that's built into Python. It's really convenient, um, [inaudible] you'll get much more in depth on these things once he's actually taught class, um, but I can say, like, "Please enter a math statement," if I can type. Okay, and the **input function** is really useful: it actually will display that on the screen and expect for the user to be putting some type of input in afterwards, right? So I enter here, I see my my statement comes up, and I can say something like, "four plus five," right? Problem is, it just returned it back to the screen as a string, "four plus five," right? Because I didn't store that into anything, okay? The **input function** is to allow you to get input from the user, um, but you have to store it somewhere, you have to save it into a variable, right? So I can use, um, something like this: I could say, "user_statement is this,", um, and it'll look however I've put it in between these quotes, right?
>
> Audio-Recorded in Discussion Section 9/21/16

Chris demonstrates the input and eval functions rather than giving an explicit definition. He models its use as a noun that can be modified by the name of the function. This is one way that "function" can be used in expert speech: one can speak of input functions, eval functions, print functions, and others. Often the word "function" is omitted, but both are indicative of fluent speech.

The TAs sometimes used the term inappropriately or used a different term when "function" would have been appropriate. For example, after demonstrating the input() and eval() functions, Chris misuses the term by describing an arithmetic operator as a function:

> Chris: So what we'll do and what we will- we'll build on, maybe, I guess like the next week or so is, uh, basically just a four operation calculator, something really simple that can do plus, divide, multiply, and the **mod function**, right?
>
> Audio-Recorded in Discussion Section 9/27/16

This was far from an isolated event. Chris consistently used the term "function" to describe operators as well as functions. The arithmetic operators are addition, subtraction, multiplication, division, floor division, and modulo (often abbreviated to "mod"). They are used in individual lines of code to complete mathematic computations. It is likely that the TAs used the word "function" in this context as a non-technical word. A function in non-programmer circles would be an activity intended for a particular purpose. In mathematical terms, a function is similar to a mathematical expression. Because many of the terms students were expected to learn were polysemous with non-technical terms they likely already knew, this potential for confusion was highly prevalent.

Anshika's students first heard the word from a student in her section, Stephen. Stephen was a white man in his second year at the university. He was a frequent participant in section and lecture, and often worked with Martha, a Latina woman who also participated frequently. At the beginning of her section, which took place in the hour before lecture on Friday, Anshika asked:

> Anshika: Uh, so can someone tell me what you've covered so far this week? What topics have you covered?
> Stephen: ((raise hand))
> Anshika: Yeah?
> Stephen: Uh, we basically just saw like how to open up IDLE, where to make your program, and then he showed us like a simple **print function** and the **function** that is supposed to be able to like [inaudible], stuff like that.
> Anshika: Okay.
>
> Audio-Recorded in Discussion Section 9/30/16

Stephen has taken up "functions" right away, providing the feedback that they know two simple examples of a function but not much else yet. Stephen refers to the print() function, which Kurt had only introduced as "print" in lecture. Kurt would speak more on functions, including print(), in the following hour. The other function Stephen mentions is input(). Anshika also mentioned functions in passing when she was describing Python and the interactive development

environment the students would be using and the existence of keywords. Keywords are identifiers that have already been assigned some meaning in Python. The names of built-in functions like print() are one type, as are operators and statement such as "if" and "in" – it is unwise to use any of these names for a new item because the functionality of the built-in item will cease. In Anshika's section, she said the word "function" four times in relatively quick succession, one of which has the polysemous meaning of "purpose" and the other three of which are the intended term:

> Anshika: So um, there's a website called python dot org where you would have downloaded "Install Python," so that website also has other information about Python. So when you actually go into writing more programs and you learn about **functions** and other stuff, so, that website will give you like a detailed list of the **functions**, a description, and how it works and the internal working.
>
> Audio-Recorded in Discussion Section 9/30/16

Anshika here does not include details about the structure or nature of functions; she only introduces them as items the students will have to become familiar with. She demonstrates that "functions" can be pluralized and spoken of as a general category. She goes on, beginning her next utterance with the caveat, "If you're interested," although the information she goes on to provide is actually required knowledge for the students, a discourse marker that might be intended to ameliorate threats to the students' negative face:

> Anshika: Also, if you're interested […] So, there is a list of keywords in your textbook. Yeah, these are a list of keywords, so these cannot be used as variable names because they ha- they have a predefined **function** in Python. So you can't use "print" because you use "print" already, it's a **function.** You can't use that as the variable's name. No, you can't use print as a variable. These are true, false, def, if, else: all of these you cannot use as variable names. And this list is in your books, so look at those, okay?
>
> Audio-Recorded in Discussion Section 9/30/16

While she does not provide content information about what functions are or how to write them, this introduction of the term constitutes modeling: students are able to hear the word used as a noun with plural morphemes, and are able to see its collocation with words like "predefined." Like Chris and the rest of the instructors, Anshika uses the word "function" to mean something other than the class term at least once during section.

After Anshika and Chris's students heard the term in section, the entire student population had it modeled for them in lecture. Kurt displayed a function on the projector screen and pointed out different items in the function, quizzing the class on the names for those items. Then he pointed to the function definition and asked:

> Kurt: Speaking of print, there's a name for this thing, too. What is it? Print? No.
> Some: **functions**
>
> Audio-Recorded in Lecture 9/30/16

Kurt's explicit direction to consider the name for the object "function" is an event of modeling metalinguistics. He does not simply describe how to use functions, but also how to talk about them. This practice appears primarily in utterances by Kurt, and less so from the TAs. Kurt goes on to explain how this new term is related to other meanings of the word "function":

> Kurt: This is a **function**, just like **functions** that you learned in math classes way back when in Algebra. You have a name of a **function**, you have some parentheses, and you have some parameter inside the parentheses, and you put a value in where the parameter is, and the **function** does something and gives you back something else, right? All right.
>
> Audio-Recorded in Lecture 9/30/16

By discussing the definitions for "function" that he expects these students already know and drawing a parallel to the new usage, Kurt is further developing the use of metalinguistics to demonstrate the meaning and grammatical features of the new term. He goes on to model the use of "function" as a modifier to another word:

Kurt: We'll talk more about the details of that. But anyway, that's a **function**, or a **function** call, or a **function** invocation, more accurately. And what this is saying is, um, "I'm gonna take this value that's in the variable us_dollars and I would like you to print it." And then Python prints it. Okay, a **function** is just a name, an identifier that is associated not with a value but with a procedure. With s- some set of instructions. And when you invoke the name of the **function**, the instructions that are associated with that name get executed. All right, so print is one such **function** that's already defined. Next week we'll start learning- I think it's end of next week- about how to define your own **functions** and use them.

Audio-Recorded in Lecture 9/30/16

Kurt repeats the term eleven times in this segment. He models its use as a singular and plural noun and shows how the word can modify the nouns "call" and "invocation," which are synonymous. He also models the interaction of the verb "define" and the noun "function," which will be an extremely common collocation for the rest of the quarter.

When he asked students to identify the displayed function, he was somewhat dissatisfied with the volume of the response. Some students knew how to identify a function when they saw it, but most remained silent. This silence constitutes feedback that the students' uptake is not yet complete. Normally when Kurt addressed a question to the class, many more students would speak up. In his next few sentences, Kurt repeated the word ten times, demonstrating its use in a function call and naming an example of a function. Kurt continued:

Kurt: Okay, all right. If you were able to answer over the past two lectures all of these questions, you're doin' great. And if you weren't able to answer these questions, "What is this, what is that, what is that, what is that?" then you are already behind, okay? And it's way too early to be behind. So catch up, all right? But for those of you who are on top of it, yay! But it's really easy to fall behind in this class, okay? As simple as it sounds right now, try to keep up now. Because it's gonna be harder and harder to keep up as we go. All right?

Audio-Recorded in Lecture 9/30/16

Kurt's statement here can be broadly characterized as a directive for those students who were unable to supply the term when prompted. He uses justification: "if you weren't able to . . . then you are already behind, okay?" and "Because it's gonna be harder and harder to keep up as we

go." Kurt also employs boosting: "it's way too early to be behind" and "it's really easy to fall behind in this class." He also employs tag questions "okay?" and "all right?" as a confirmation check, to signal his expectation that students are listening and are in agreement. Because these tag questions appear in the context of a directive, they are also a request for intent to comply. Rather than use the inclusive "we," a common strategy among the native speaker instructors, Kurt chooses the second person plural, directly addressing the students for increased directness.

Kurt's initial modeling included the use of "invocation" with the target term, a word many students did not recognize. One such student, who was not a participant in this study, asked what an invocation was, so he defined it for them:

> Kurt: Uh, it's, um, to inv- to invoke the name of something, to call on something: "Oh, Satan! Strike down all Windows computers!" It's something like that, you know?
> Class: ((laugh))
> Kurt: Look, I'm looking out for you guys. Um, no, but it's- it's- it's- when we- we ask something to do something, we sort of invoke its powers. And so it's just a technical term for that, is function invocation.
>
> Audio-Recorded in Lecture 9/30/16

Kurt uses the synonym "call" to strengthen the collocation of the two words, and he also evokes the non-technical meaning of "invoke" - to call on a deity to perform an action. This strategy harnesses a potentially pre-existing lexical item in the students' minds. If they knew the word "invoke" before, they can now slightly modify its definition to include functions alongside deities. Kurt also uses humor as a distance-reducing device and as a mnemonic mechanism: if the students can remember the joke about Satan striking down Windows computers (which Kurt has by now established he has trouble with), perhaps they can remember that invoking a function means asking it to do something. Humor is a part of the personal/social discourses programmers are socialized into. The joke allows the students to do social referencing, understanding that a joke has been told and therefore licensing them to laugh; the students' laughter also constitutes

feedback in the form of an affective stance: they are joyfully incorporating this word into their lexicon.

The relationship between functions, variables, identifiers, is one of the learning objectives of this lecture. An identifier is the name given to functions, variables, and other objects. This can give rise to confusion because a person could equally say "This function is print()" and "The identifier of this function is print()." In the activity at hand, Kurt had pointed to functions and variables to ask the students to name them, and the target answer had been "identifier." Students are given opportunities to ask and answer questions, supporting professional discourses that allow speech opportunities for participants. Katherine gives feedback by asking a question about this:

> Katherine: Uh, just now when you highlighted print and you said it was an identifier, can you just explain? Because I thought it was a function.
> Kurt: But functions have names, and names are identifiers.
> Katherine: Oh.
> Kurt: And variables have names, and so names are identifiers. So any variable we have, that thing that it's called by is also an identifier. And any function that we have, and that name it's called by is called an identifier. Does that make sense?
> Katherine: Yeah.
> Kurt: Okay, good. Good question.
>
> Audio-Recorded in Lecture 9/30/16

Kurt takes this opportunity to strengthen the conceptual connection between "identifier" and "name" - this way he can ensure that the difference between an identifier and a function or variable is clear. Kurt's use of repetition allows the appropriate use of these terms to become evident. Variables and functions "have" identifiers; it is not appropriate to say that they "are" identifiers. They are not identifiers: names are. Kurt also provides a confirmation check: "does that make sense?" This is not a tag question provided as a discourse marker, but a question for which he wishes to elicit a response. Katherine feels satisfied, so she answers "yeah" as feedback, and Kurt is prepared to move on.

In order to model the use of and language surrounding functions, Kurt demonstrated the use of the input() function. To call or invoke this function, one must type the word "input," followed by an open parenthesis, then type in something that will serve as a prompt, such as, "Type your name here: " Once the user types a closed parenthesis, that prompt will appear in the shell, followed by a blinking cursor waiting for the user to type something. He demonstrated the ways input() can be used to get typed input from a user and use that input for some purpose. Next, Kurt defined a function called marvelous() that displays the prompt, "enter your name: " and takes an input from the user, then prints a sentence using that input: "Hello [name], you look marvelous!" Rather than model the oral use of "define," he modeled typing a function definition. Kurt did not wish to present function definitions as a current skill, so he chose not to emphasize it.

One potential area for confusion was that there exists another word that can be considered partially synonymous with "function." The built-in Python functions can also be referred to as statements, leading to utterances such as this one:

> Kurt: But if there's not direct user interaction, there's not like an **input statement** or a **print statement** or stuff like that in the function, and it's just, "Add values to a parameter, parameters are wrong, let it explode." Okay?
>
> Audio-Recorded in Lecture 10/05/16

While using the word "statement" in this context is correct usage, it could be considered a missed opportunity to model the use of "function" or explain the appropriateness of using both terms. The term "statement" can also be applied to "return," as in "a return statement." This could prove problematic as return is not a function; however, students never expressed or demonstrated confusion about the term "statement."

In the winter, Chris was the first person to use the term. he did so in the context of explaining the difference between a shell in an interactive development environment and a saved file.

> Chris: This is, um, kind of more than what you guys are gonna do, but right now, but all- I can make a **function** here inside of this, right? Um, which is cool. I can just say, "Return, [inaudible]", take it, and that works. Right? So now if I call add with two numbers, it works!
>
> Audio-Recorded in Discussion Section 1/18/17

Chris identified this first use of the term "function" as being a bit ahead of its intended introduction. He typed a very short function with a return statement, using techniques and terms the students had not encountered. As he did so, he used the tag question "right" to solicit confirmation from the students. By saying, "This is, um, kind of more than what you guys are gonna do," Chris gave the students permission not to attend to this modeling event. Instead, Chris's main concern in this moment was demonstrating what "worked" and what did not in the shell as opposed to a file. He continued, avoiding the use of the term as he did so:

> Chris: Um, but here's the problem with that, is that when I close out of this and I come back to it, right? [inaudible] So you remember I called it "add" and it added two variables, right? And that worked the last time I did it. It doesn't work anymore. Right? That's because this is just an interactive version of Python. It will store whatever you're currently doing in your session, but it's not going to store those things over time. […] Um, but the standard that we get with Python I.D.L.E. is also really nice.
>
> Audio-Recorded in Discussion Section 1/18/17

Again, the primary discussion concerned the idea of functions "working" or not working, not the meaning and use of "function." As such, the term was ignored until Chris needed it again to demonstrate how the function he'd written was meant to work:

> Chris: Um, so we can do that same thing that we did before, where we made that **function** here, right? And we have **function** with two parameters, it's going to return a value just like it did. And it's just gonna return the sum of those. Right? So this is pretty, hang on, pretty basic two-line **function**. I don't know if he's

> gotten to **functions** yet in class, um, but this is pretty basic. x and y we know are variables, um, add is going to be the name of our **function**.
>
> Audio-Recorded in Discussion Section 1/18/17

Chris's modeling of the term included repetition (five uses) and the addition of the plural -s morpheme. Students were looking at the function he had written projected on the board, so they could see what a function tended to look like. By calling it a "pretty basic two-line function," Chris implies to the students that normal functions will have more lines than that. He also demonstrates the use of "return" as a verb that explains what the result of a function will be. Next, Chris demonstrates the use of "function" to modify "head", a term he does not explicitly define but whose component parts he shows on the projected screen:

> Chris: When we're creating a **function** head, we know to add that def in there. He'll go over that in lecture with you guys. Um, and then all we're doing here is just returning the sum of x and y. Right? This is really basic. It's pretty much a useless **function**. We would never write any code like this. But here we have it. We'll use this to show you guys how to save any kind of **function** here.
>
> Audio-Recorded in Discussion Section 1/18/17

The use of functions in this process was not a strategy Chris or the other instructors employed to demonstrate the difference between the shell and a file, or to show how to save a file, in the previous quarter. Nor did any other instructors do so in winter. It requires Chris to model the use of "function" alongside words the students have not heard yet, such as "parameters," "return," and "head." This high concentration of new terms is quite dissimilar to Kurt's strategy, described above. Additionally, the students were not given many chances to provide feedback regarding their sense of how the word should be used. When, at the end of section, Chris asked for questions, none pertained to functions, so students did not volunteer feedback or uptake at that time.

In winter, Kurt used the term before he had planned to because of a student's incorrect answer to a question. Kurt had pointed to a group of variables and a print function on the projected screen, hoping to hear the word "identifier" from the crowd. This is an example of the professional discourse of scaffolding:

> Kurt: Now, this one's kind of a trick question. Hm? Because I already asked you earlier, what are most of- you know, what are all the things in green? Um, and you all said they were variables. And that was right. But now I've got an extra thing in there that's in green. They're all variables except for one thing. So they're not variables, they're sort of a larger, more encompassing category, and who knows what the name is for all the things that are in green right now? Anybody? Again it's a n- it's a name we've already mentioned once in slides, two. Yes?
> [non-participant]
> Kurt: Those are not **functions**! 'Kay? That one right there is a **function**.
> […]
>
> Audio-Recorded in Lecture 1/18/17

By incorrectly calling the items "functions," the student has provided feedback that they are not yet comfortable with the term. Kurt does not ratify this use of the term, then he points to the print function when the student says the word, and names it. He comes back to the definition and use of the term before he had planned because of this premature modeling event:

> For example, oh- it says we'll talk about those in a little bit, but I'll actually talk about them now because somebody pointed it out. That thing right there, "print," that's not a variable name. That's the name of a **function**. It's a **function** that's supplied by Python for us to use that allows us to easily print stuff on our display. If we didn't have the named **function**, "print," if we had to write all the Python instructions just to get something to print on the display, i- it would just overwhelm us. There's just so much stuff. Getting stuff printed is not- is not necessarily easy. So, rather than have us rewrite print **functions** over and over and over again, Python people just gave it a name and said, "Use print, and that'll get stuff on your display that you want printed." We'll get back to that in just a second, too.
>
> Audio-Recorded in Lecture 1/18/17

He uses it four times, once in conjunction with the modifier "print," to establish an association between functions and procedures. He uses inclusive "we" as a distance-reducing strategy, modeling appropriate expert talk as he does so. Finally, at the pre-planned time in the lecture,

Kurt describes functions once more using a function he wrote that converts an amount of U.S. dollars to its value in Canadian dollars:

> Kurt: And then, last but not least, I think this is last on our deconstruction of the program, this is a **function**. Um, print is the name of the **function**, again, that's Python folks have given us to print stuff out. US Dollars is a variable that, in this particular case, is the product of a thousand and point seventy six, so, uh, when this is all done US Dollars is gonna have, uh, seven hundred and sixty one dollars and twenty cents assigned to it, seven sixty one point two. Uh, and that print is the- is the- the name of a **function** on- sometimes we call it a **function** call or a **function** invocation as if we invoke the **function** like we invoke a magic spell or something like that. Okay? Um, **function** is just a name for a col- not for a- a number, but for a collection of instructions. And, uh, every time we call on that **function**, then it does the same thing for us, and, in order- most **functions** will expect some sort of thing that we put in these parentheses, like, "What do you want to print?" "Oh, I want to print the value that's in US Dollars." All right? So that's called the argument to the **function**. Sometimes we call it the parameter of the **function**; we'll explain all those different things, too. Hm. Um, so, uh, very soon, we'll show you how to define our own **functions**, so you can put them in your program. In the meantime, uh, we have some existing ones that you can use, like print. And like we said, **functions** typically expect one or more of these values to be passed as its parameters. 'Kay, we think of that as the input to the **function**. All right? Where am I here? Okay! If you were able to answer all the questions as I ask them, then you've done your reading and you are good people.
> Class: ((laugh))
>
> Audio-Recorded in Lecture 1/18/17

As in the fall, Kurt models the use of function as a plural and singular noun and its role as a modifier for "call" and "invocation." He models its proper use as the recipient of the verb "define" but not "call." In describing the function, Kurt also models a discursive practice, which is to personify functions. He demonstrates that functions "expect" parameters, which Kurt and the TAs all agree is appropriate. After this explanation, Kurt gave a similar warning he gave in the fall: at this point the questions identifying pieces of the displayed function should have been easy for the students. If they were not, students needed to catch up. Again, he jokingly connected this academic failure to a moral one, calling those students who were prepared "good people."

When introducing students to the term "function," Kurt was careful to repeat the word many times and model its use in several grammatical roles. He also demonstrated its relationship to other class terms in order to model its proper integration into spoken English as used by programmers. For some students, Kurt's practiced speech and activity were their first aural introduction to the word; however, for some it was not. The TAs were not as polished or fluent in their modeling practices, so their students' first exposure to the term was characterized by potentially imperfect usage and a deluge of unfamiliar terms, which could add to any existing trepidation concerning their mental fitness for the topic of computer science in general.

Student uptake of "function" and related words

Students needed to be able to use the term "function" almost immediately after it was modeled for them, in order to ask for help from their instructors. One concern raised by researchers interested in students asking for help is that the power differential between a student and an instructor could lead to increased face threat. Students need to be able to participate in the model for Language Socialization: modeling, feedback, and uptake. To demonstrate feedback and uptake, they must risk exposing themselves as less knowledgeable than they would like to be. Students may be overly concerned with preserving their teachers' negative face, and might ask abbreviated versions of their questions or fail to ask for additional information if the first answer was insufficient. Teachers might also be concerned for their students' positive face and avoid correcting them if they perceive that the student might lose confidence. In this section, I compare the students' use of "function" in questions to instructors to their use among peers. With instructors, students who were not yet comfortable using the term tended to employ silence or deictic words to avoid saying it, causing the instructors to fill in the blanks.

In section and lecture, students were asked to use the terminology in the course of reviewing information. When this happened, instructors tended to provide context and surrounding terminology, so that students could fill in the target terms without having to form full expert-fluency sentences. For example, in section early in fall quarter, Anshika provided a speech opportunity by asking the students to provide the instructions for defining a new function, using scaffolding to make it possible:

> Anshika: So uh, the first step in writing a function: what do I do first? (.) What is the first thing I do? (.) I use a keyword. What keyword is that? (.) Okay, you've all written functions, so what keyword do I start with?
> Tim: D.E.F.
>
> Audio-Recorded in Anshika's Section 10/7/16

When Tim supplies the correct answer "D.E.F.," he speaks the individual letters. A programmer would just say, "def," the first syllable of "define," to sound like the English word "deaf." Anshika repeats the word using the proper pronunciation in the course of ratifying Tim's answer:

> Anshika: Def, yes, so, okay.
>
> Audio-Recorded in Anshika's Section 10/7/16

By saying "Def, yes, so, okay," Anshika corrects the smaller error -- the pronunciation -- while acknowledging that the answer was correct, thereby reducing the potential threat to Tim's positive face in the course of a correction. As they continue, Anshika asks leading questions for which Tim can supply single-word answers or simple noun phrases:

> Anshika: So what do I- what follows after that?
> Tim: space
> Martha: I'm confused.
> Anshika: Okay, a space, and then?
> Tim: The function's name.
> ((Martha and Stephen whisper to each other))
> Anshika: Yes, so the function name. What next?
> Tim: Parameters.
> Anshika: Yeah, so how do I enclose my parameters?
> Tim: Parentheses.

Anshika: Okay, so do this one and then I can have- okay or how many i have and [inaudible]. So what after this?
Tim: Colon.
Anshika: Yes, a colon. So this this is how you start a function name, and then when you- then you have your body of the function here, mhm.

Audio-Recorded in Anshika's Section 10/7/16

In each conversational turn, Anshika ratifies Tim's answer and requests another. This practice, which took place in the sections and in lectures, creates an environment for students to begin using the terminology before they are ready to generate complete sentences, and allows the instructors to model the correct use of the terms in context. Anshika's methods here are also indicative of some missed opportunities within the uptake-support practice. Each time instructors accept a correct answer, a more language-socialization-supportive practice would be to rephrase the single-word answer as a full sentence. For example, when Anshika asks, "So how do I enclose my parameters?" and Tim says, "Parentheses," Anshika could answer, "Yes, in a function definition, parameters are written in parentheses after the function identifier" rather than "Okay." As a conversational turn it may feel repetitive or unnecessary, but as a practice for supporting language socialization, it provides more opportunities for modeling the integration of the relevant terms into the students' language. In the meantime, Martha's feedback, "Wait, I'm confused," is an epistemic stance: she is unsure why the correct answer is correct. Her concerns are addressed by Stephen rather than Anshika. By ignoring Martha's feedback and choosing not to engage with her, Anshika may be reinforcing the practice of avoiding such feedback in the classroom setting.

Students in lab had their computer screens available as a mechanism for avoiding using the terms they had learned. For example, in lab toward the beginning of fall quarter, Travis wished to ask a clarifying question about docstrings. Docstrings are text that is written just below

a function's head (the function definition). When someone calls the function or looks it up using help(), that text will appear. Students were expected to write a docstring for each function they defined that would inform the function's user of its purpose, which parameters it expected, and what the result would be. Travis was able to ask his question without the use of the term "function" because he was pointing to a function he had written on his screen:

> Travis: So in lecture, he **did** something like this. ((point to screen)) Do you know what that is?
> John: That's a, uh, docstring.
> Travis: Docstring?
> John: Yeah.
> Travis: So what does that do?
> John: It will like explain to you like what this, uh, function is about. Like what it says. You can use, like, help and add the function name where this will show.
> Travis: Okay so, like, if I were to like write **this** into the shell and then I did the parentheses-
> John: Oh don't- don't do anything in the shell.
> Travis: Oh, okay.
> John: Unless you're running uh, a function or something like that, don't do anything else in the shell.
> Travis: Okay.
>
> Audio-Recorded in John's Lab 10/7/16

Rather than use the modeled terminology, Travis uses gesture and deixis to ask his question. He uses the pro-verb "did" and the deictic pronoun "this," effectively avoiding the target words. A more fluent question might have been, "In lecture, he typed a string below the function definition. Do you know what that is?" Next, he uses deixis again for his question, "If I were to write **this** in the shell…" where "this" refers to a function call. He is not able to finish his question because of a misunderstanding on John's part that causes him to interrupt.

Another interesting challenge presents itself in this example, one which took place several times. The instructors were told that students struggle to differentiate the shell from a file. Code should be written in a file, and only tested in the shell. The shell has some features that allow programmers to write code in it, but that code cannot be re-used or sent to others. For this

reason, students must learn early on how to write in the file, not the shell. In this case, Travis is asking about an appropriate activity for the shell, calling a function (calling a function allows programmers to test it). He is asking for verification of his belief that docstrings will appear if one calls a function in that environment. John, however, is concerned that he wishes to define a function in the shell, a mistake he has been warned students will want to commit. For John, the teaching objective "stop students from coding in the shell" supersedes his understanding of the question he is being asked. It is so urgent to him to stop Travis from coding in the shell that he interrupts him and delivers a bald on-record command: "Oh, don't do anything in the shell." Travis agrees to this twice, but does not correct his misunderstanding or re-broach his original question. It is possible that the teacher-student power differential caused him to fear losing positive face by continuing to express confusion, or that it caused him to fear threatening John's negative face by asking for more of his attention.

Even students who were highly involved in the class and performed well on assignments tended to use gesture or deixis to avoid the terms associated with "function." For example, in Anshika's section, Stephen volunteered information intended to help Anshika understand what the class had done in lectures the previous week. Stephen was able to use the term "function" but not its appropriate collocated terms:

> Anshika: Okay. So this is a generation of while loops[2].
> Stephen: So it was the [inaudible] to loop random numbers, we **did** a coin toss function, a break statement, and then sequences to see how to count the N's.
> Anshika: Okay, good. Okay, so uh, all- you must know by now the basic syntax of a while loop statement.
>
> Audio-Recorded in Section 10/25/16

Again, the pro-form DO stands in for the preferred verb, "define." Stephen is comfortable enough using the word "function," but not its accompanying terms. In the service of moving the

[2] A while loop allows a programmer to execute a block of code over and over as long as some condition is met. The programmer sets that condition at the beginning of the loop.

class along, Anshika misses a chance to ask him to use those terms, instead ratifying his choice of words. As in John's lab, an important teaching objective prevented a chance at language socialization.

In lab, instructors often required no use of class terminology, or even a general description of the problem they were experiencing. In Chris's lab during the third week of instruction, Rheanna was partly finished writing a sequence of functions that would calculate a user's body mass index. Chris sat next to Rheanna, and they began their conversation this way:

> Rheanna: Like, I just- I can't get it. Like-
> Chris: Sure. All right. We've got (.) okay.
>
> Audio-Recorded in Lab 10/11/16

Rheanna's request for help is phrased as a problem statement: "I can't get it." This statement is accompanied by a gesture to the code she has written, alongside a shell with an error message in it. A problem statement can be understood as a request for help without specifying the particular type of help she needs. Chris ratifies this as a request for help and begins describing the route to a solution.

> Chris: So you actually have everything lined up exactly how you want it to be. We just gotta change a couple things around. Right?
> Rheanna: 'Kay.
> Chris: So we want this body mass function to do everything that you've written right here, right?
> Rheanna: Mhm.
>
> Audio-Recorded in Lab 10/11/16

He reduces the potential face threat involved in correcting a student, using inclusive "we," mitigation ("just," "a couple things"), and checking for validation ("right?"). Chris's help, however, entails providing all the answers to Rheanna and requiring only that she agree with his solutions rather than participating in scaffolding that allows her to provide class terms.

> Chris: We want it to take in a name, we want it to get meters, we want it to get kilograms, and then we want it to, uh, print what the body mass index is and figure out those

values, right? And we want to use these functions that we've already written up here to be able to do that.
Rheanna: Yeah.
Chris: Right?

Audio-Recorded in Lab 10/11/16

Rheanna is never required to explain the logic behind these solutions herself, nor must she use any of the course terms that Chris is providing. Furthermore, Chris takes the keyboard from Rheanna and begins typing into her assignment:

Chris: So I'm gonna take out- I'm actually just gonna move these around because we don't actually need these right now, 'kay? So I'm gonna take those out (.) and that's just gonna get rid of- 'cause like that was- you're tryin' to call B.M.I., right?
Rheanna: Yeah.
Chris: And you're giving it me- meters and kilograms, which you're gonna get down here.
Rheanna: Okay.
Chris: And then what we want for these ones, and I think you can look at the way this is- the homework is written, is these first three, I'm writing a function that's going to take in a parameter and return a value, right? Problem one, problem two, and problem three, that's what they all do. They all take in a parameter, right, and they all return a value.
Rheanna: Mhm.
[...]

Audio-Recorded in Lab 10/11/16

Chris has control of Rheanna's computer for several minutes, copying and pasting her code in different areas, as well as typing and deleting lines. He is the only one to use class terms.

Chris: So it's gonna look over here, and- well, actually, I'll write this one for you and then you can do the other one the same kind of- we'll see if you can (.) do it yourself. Right? So I'm gonna call your function, inches to meters, that you created up here. Right?
Rheanna: Mhm.
Chris: Because I want my meters to be- er, my inches to actually be meters. And I'm gonna give it- here we called it meters, so it looks a little confusing, but I'm gonna give it meters which we know i- is actually in inches right now. Right?
Rheanna: Okay.
Chris: So now what this is gonna do, is this is gonna take whatever value the user gave it for meters, right?
Rheanna: And plug it there-
Chris: Um, it's gonna plug it in up here, exactly. So it's gonna convert that and it's gonna return the result, and now it's gonna re-store it back into meters.
[...]

Chris: 'Kay? So now I have it in meters. I can do the same thing with kilograms, and then I can do the same thing with (.) and we'll create a new variable down here, um, so I'll let you do the kilograms one yourself, and then we can do a new variable after that. [inaudible].

Audio-Recorded in Lab 10/11/16

In the course of having her questions answered, Rheanna has a moment of confusion and asks the beginning of a clarifying question, but then negates that question once she realizes that she had been wrong to think the result was pounds:

Rheanna: Hm. (.) Mm. (.) Was it- is it pounds- no, kilograms.
Chris: So it's gonna be kilograms, right. So- and what the variable name is up here and what the variable name is down here don't actually matter, right? So you have to use it in the correct scope no matter where you are. Okay? So here you've called it pounds, so we use pounds here, right?
Rheanna: Mhm.
Chris: But here, because we've stored in the kilograms, we have to send kilograms to it- this kilograms actually becomes this pounds, right?
Rheanna: Yeah.
Chris: So that's kinda how that works.
Rheanna: Okay. So then- that [inaudible] that makes a lot of sense.
Chris: All right.
Rheanna: 'Cause I was like, "How are you gonna relate this part-"
Chris: Yeah.
Rheanna: "-to what we did before?"
Chris: Right! Right.
Rheanna: Yeah. ((laugh))

Audio-Recorded in Lab 10/11/16

At the end, Rheanna concludes their conversation by claiming that the result she has reached "makes a lot of sense." At this point, much of the code on her screen has been written by Chris, not her, and she has been watching him type and back-channeling agreement as he explains himself. She has participated in a long and detailed conversation that results in her function's successful completion without having to utter a class term or originate a plan for the design of her function.

Pair Programming and Student Uptake

In the pair programming exercises, introduced in the winter quarter discussion sections, students were required to direct one another in writing snippets of code to solve problem sets. One student took on the role of "navigator": they told the other student what to type. The other took on the role of "driver": they typed what the navigator dictated to them. This practice achieved two objectives: students had to produce full phrases and sentences involving course terminology, and they had to recall procedures they had learned without expert intervention. In this example, Sang and Maya are writing a function for the first time:

> Sang: Uh, I think, you try doing the **function** first?
> Maya: Like def **function** first?
> Sang: Yeah. Def **function**, and then, um, [inaudible] the name of- I found out that we might need (.)
> Maya: Parentheses?
> Sang: Parentheses. And then, that's why it's, um, okay. So, ((laughs)), um, okay.
>
> Audio-Recorded in Chris's Section 02/08/17

Sang uses the verb DO rather than "define" or its acceptable abbreviation "def," but rather than allow this avoidance move to stand, Maya requires clarification. As feedback, she rephrases Sang's suggestion exactly, except that she substitutes "def" for "doing," thereby adding detail to both their understanding of the proper use of "function." Sang ratifies and takes up her suggested substitution, repeating, "Yeah. Def function," creating a new memory for each of them of both hearing and using words associated with the target term. Ultimately, the omission of a determiner "a" or "the" would be considered novice-like usage, but the students' agreement that the use of "def" is preferable to DO in the context of a function suggests that they are developing in the direction of expert terminology use.

In lab or after class, students often asked for help with functions they had written. They were able to achieve this end without having to use many of the desired terms, by the use of

gesture and deixis. This practice of avoiding class terms was supported by instructors who often supplied those terms themselves, either by answering the student's intended question without requiring them to ask it using expert speech, or by typing the correct code on the student's laptop. Without the instructors' help, students in pair programming activities were required to generate the proper use of the terms themselves, with only each other to accept or reject that use. To the second learning objective, recalling programming procedures, the students discussed the solution among themselves and used the computer to test whether their understanding was correct. In this case, neither student took the position of expert, but by consulting the computer they were able to socialize one another and themselves without requiring one. In a pair programming activity between Brigitte and Taj in John's section, the pair needed to write a function that would prompt the user to type in a name, count the number of characters in that name, and then print one of two strings depending on the number of characters. Brigitte begins by describing the elements they will need to include in the function:

> Brigitte: Mm- so we have to find a way, where like, we can take the letters of the name,
> Taj: Mhm.
> Brigitte: and then they like-
> Taj: Yeah, and then they print there.
> Brigitte: Yeah. Um, [inaudible].
>
> Audio-Recorded in John's Section 02/06/17

As the navigator, Brigitte must design the function. She has identified one of the three necessary elements: the function must take in the characters of the user-entered text. Taj, while he is not the navigator, supplies the third: it must print a response. The second element, counting the number of characters in the input, is the most recently learned skill and will require some negotiation. For Taj, the driver, supplying directions is technically outside the scope of his role. Drivers often took on the responsibilities of navigators during the pair programming activities. The purpose of

the roles was to ensure that the two participants collaborated on a single product, rather than working side by side on two separate solutions. Their practice of taking on one another's roles actually increased their ability to socialize one another by allowing them to negotiate meanings.

Brigitte uses the verb "take" to refer to the function taking input from the user, an expert-like choice of words. She calls the characters "letters," however, a practice that marks her as a novice. "Letter" is not a term in Python; functions that involve working with letters are more accurately described as involving "characters," a category that includes letters, punctuation, and spaces. This misuse of terminology is not addressed in this conversation. As they continue to work together, Brigitte and Taj must discover that they need to use len() to achieve the second element of their function, counting the characters in the string entered by the user:

> Brigitte: How would we do that?
> Taj: name_judge, that's weird ((laughs)).
> Brigitte: Yeah because we have to find a way where, like, name actually becomes- I think maybe there's a function that like takes the letters.
> Taj: Maybe int() function?
> Brigitte: Ah?
> Taj: int(), maybe.
> Brigitte: int()? Like this?
> Taj: Yeah. Then I think that should work. Um.
> Caitlin: I think it's time to switch roles, so if you were driving you should navigate, if you were navigating you should drive.
>
> Audio-Recorded in John's Section 02/06/17

Initially, Brigitte poses the question: "We have to find a way where, like, name actually becomes…" In doing so, she does not quite achieve a complete formulation of the issue at hand. She has successfully taken the user's input, stored it in a variable called "name," but now she must devise a method for converting "name" into an integer that describes the number of characters in the string that was assigned to it. It is not necessary that this integer also be stored in "name," but Brigitte's question suggests that she believes it does. Either way, Brigitte and Taj must call on a function that will do the counting. Distracted by the knowledge that the desired

output of the function they need is an integer, they focus on the int() function, which converts a number of any type into an integer. Because the object they need is a string containing characters that are likely not numbers, using int() will result in an error. Taj suggests int() using mitigation in the form of the adverb "maybe" and a high-rising terminal, implying a question rather than a declaration. Brigitte's feedback is an epistemic stance: she is unsure, asking simply, "Ah?" in order to avoid endorsing or rejecting his suggestion, and Taj repeats his suggestion with the same mitigating adverb. Because there is no expert present to tell them whether int() is the correct function, Brigitte types it in and checks with Taj to see whether she has adequately performed his suggestion. He agrees that she has and posits his prediction that it will work, and they attempt to run the function as written. When they receive an error, they react:

> Taj: Yeah, so I- um, wait, okay. Whoa ((laughs)).
> Brigitte: ((laughs)) No.
> Taj: Yeah, so int() is wrong.
>
> Audio-Recorded in John's Section 02/06/17

The laptop serves as a tool to provide the students with guidance without too much influence. Taj made a suggestion, Brigitte executed it, and they discovered that the suggestion did not work. Socialization events like these were relatively common in pair programming activities, but were very rare in the presence of instructors. An instructor would likely have told Taj not to use int() because it did not do what they needed, and the joyful experience of error above could not have taken place. Taj and Brigitte have chosen the wrong course of action by using int(), which they discover thanks to a bright red sequence of text displayed on their screen: an error. This display prompts laughter and exclamation of surprise from both students, followed by a rejection of the use of int() and a conversation about what the right answer would be:

> Taj: Um, maybe s() here?
> Brigitte: There- 'cause there's like, there's a function,
> Taj: Mhm?

Brigitte: that takes how many letters there are in the name. Like,
Taj: len()?
Brigitte: Ah- yeah, I- yeah, yeah!
Taj: Yeah ((laughs)) yeah, let's do that. I don't know if this is the way you write it? [inaudible]. Aw, yay! ((laughs))
Brigitte: Yay! ((laughs)) Now try the other one. Um, yeah, that's it! We're done, we're good.
Taj: Yeah!

Audio-Recorded in John's Section 02/06/17

Taj has another incorrect idea: using s(), which is not a built-in function in Python. It is possible he is thinking of str(), which converts an item of any type into a string. This is not what they wish to do. Brigitte reframes the goal once more: "there's a function that takes how many letters there are in the name." Neither Taj nor Brigitte corrects her use of "letters" rather than "characters." This re-focusing move reminds Taj of the function they need: len(), which takes in a string and returns an integer representing the number of characters in that string. Brigitte expresses an affective stance: excitement that she recognizes this as the correct answer, repeating "yeah!" and watching as Taj types in the function call. When it works, both students exclaim, "yay!" and celebrate their success. Together, the two socially reference each other and license each other to express joy in a moment of silliness.

An additional feature of introducing pair programming in the second quarter was that lab hours in the two quarters unfolded somewhat differently. In Fall Quarter, most students did not speak to each other unless they had been friends before enrolling in the course. At the end of the quarter, when lab hours were extremely full, students began to form small groups. In Spring Quarter, students formed groups in lab within approximately a month of beginning the course. These students spoke to each other about their code, gave and asked for advice, and spoke on topics not related to the course. The difference in social behavior I observed could be the result of the increased requirement to interact, but it could also be due at least in part to the fact that

students in the second quarter may have felt more confidence to speak with students unknown to them due to increased experience participating in university activities.

Pair programming provided an environment in which little to no power differential existed, creating a space for students to negotiate meaning and make mistakes with less fear. Conversations between students were characterized by more emotive speech, higher volume, more pitch variation, and more agreement than conversations between students and instructors. Students spent more time engaging with possible solutions to problems, requiring them to explore class concepts more fully. Students were not able to model ideal usage of the terminology, sometimes supporting one another in incorrect use, but in some cases they were able to influence one another to increase their likelihood of using class terms.

Index

The term "index" refers to an integer written in square brackets that refers to an item's place in a sequence. Sequences are strings, lists, and other objects that can contain multiple items; however, in this course, indices were mainly used in the context of lists. In a list named "mylist" containing the elements ['x', 'y', 'z'], typing mylist[0] would return 'x', typing mylist[1] would return 'y', and typing mylist[2] would return 'z'. Requesting the length of that list by typing len(mylist) would return an integer, 3. As is the case in many programming languages, the first element is referred to as element zero, rather than element one. An element in a list can be changed by assigning a new value to its index. Typing mylist[1] = 'a' would cause the list to read ['x', 'a', 'z']. This cannot be done with strings, however. Based on conversations with Kurt and the TAs, this distinction was considered a potential point of confusion for the

students. The term can be used as a noun or a verb, but was demonstrated in most cases as a noun and sometimes a nominalized verb, as in "Have you done indexing yet?"

This term was introduced later than "function," approximately halfway through the course. While Kurt took a similar approach to modeling this term as to modeling "function," using it several times within relatively few sentences, the TAs tended not to prioritize this modeling method. As in the case of "function," some students heard the term in discussion sections before it was introduced in lecture. While indices can be used to access characters in a string, and strings were taught before lists, Kurt introduced them only when it was time to use lists.

Modeling "index"

In the fall, Anshika spoke the word "index" first. In the winter, it was Chris who did so. Anshika initially used indices to demonstrate the use of strings, which are sequences of characters. In demonstrating the use of indices, Anshika misses several opportunities to model its use as a term. In the following example, Anshika is doing live-coding, demonstrating the behavior of her code in the Python shell. Her primary objective in this section is to describe strings and their uses to her students, not to explain indexes. This may explain why she was not focused on modeling the term in as much volume as she might have been. However, when the primary objective was to learn how to use indexes, she and the other TAs continued to avoid the term "index" at several opportunities. The target term "index" is represented in bold, as are moments in which Anshika used non-target terms where "index" might have been appropriate. The null sign [∅] appears where Anshika said nothing but "index" might have been appropriate.:

> Anshika: And also in strings, you can access a particular character from a string. So that **access** I use, using these square brackets, and inside the square brackets, **it** starts

> from zero. So if I have a string, "abcd", A is at the **point** zero, and then one, two, three. So if I **do** string of zero, it will print A. (.) From s two, at the one **position**, you will have J. So you can access individual characters in the string also using **indexing** from there, zero and one, and you can also access specific substrings from inside. Where, say start, (.) So, here I'm saying I want characters from string **[⌀]** one, from **index** one, up to and not including **index** three. So I have A, B, C, D, so from **index** one B, and then C, is **[⌀]** two, B is **[⌀]** three, and, but I won't include it here, so it'd be up to and including three, it's B, C, so you can also extract substrings like this one. [...] So this is how you access individual characters and then get substrings from the string too.
>
> Audio-Recorded in Section 10/21/16

In her first sentence, Anshika models a desired collocating verb to "index," which is "access." Indexing is a way to access elements of a sequence. In the second sentence, "access" perseveres in a slight misuse, resulting in the use of "access" as a noun where "index" would have been more appropriate: "that access I use" is not a common phrase in this community. Anshika is demonstrating the use of indexing to access characters in a string, primarily using the first person singular as a cue that she is modeling expert practices. However, the inconsistent use of terminology is likely distracting. A few weeks later, Anshika uses "index" much more consistently, but only as a verb:

> Anshika: What can- I can- (.) ((cough)) access a list element by **indexing** it this way. (.) So again the **indexing** begins from zero. So zero, one, two, three. So similarly for the string (.), I could **index** it using zero and one. So in this place, strings and lists are similar.
>
> Audio-Recorded in Section 11/4/16

Still modeling the use of the term using the first person singular "I," Anshika does not choose to live code this example. This may be because she believes her students are already familiar with the practice of indexing elements of a string and expects them to extend this practice to indexing elements of a list. The use of "index" three times in as many sentences constitutes a stronger modeling practice, as does pairing it with proper collocations such as

"access," "list," and "element." The use of "index" as a noun is missing in this case, however. TAs tended to prefer either the noun form or the verb form, rather than using both equally.

While indexing is relevant to strings, Kurt did not intend to introduce the concept until the fifth week, when the students were to learn about lists. To demonstrate this in both quarters, Kurt loaded in a list of lists, each one containing names and other details about famous mountains. Because this took place toward the end of a lecture, he only began to describe the concept, but left demonstrating it for a later date. In this example, Kurt points to the different elements of his list and describes them:

> Kurt: Here's a mountain name, here's a latitude, here's a longitude, here's, uh, height. Um, and we don't think of things as changing a whole lot. Mountains don't typically, you know, grow or shrink. Actually they do, but that's a different issue for a geology class. Um, but, um, you know, we could also use the- the fact that, um, these individual elements in a list are- are accessible through an integer **index**. We could treat each one of those **slots** in a list as a- as a separate variable to be used in whatever way we want to, in such a way that they change over time.
>
> Audio-Recorded in Lecture 11/4/16

Because this is a preliminary introduction to indexing, Kurt only uses the term once. He does, however, miss an opportunity to use it again, saying "one of those slots" instead of "one of those indexes. In a later lecture, Kurt would introduce "index" in more detail, with more involvement from the students. In this example from winter quarter, Kurt describes but does not demonstrate indexing items in a list:

> Kurt: And the for loop is designed to work with sequences. What's a sequence? Nobody knows! That's good! Excellent. That- that puts you in good shape for Monday's exam. Alright.
>
> Audio-Recorded in Lecture 2/22/17

In preparation for explaining how indexes work, Kurt uses a relatively new term, "sequence." While "what's a sequence?" could be interpreted as a discourse marker and not a solicitation of

answers, Kurt evaluates the students' non-response as a failure to answer a question. His sarcastic joke afterward is less a distance-minimizing move and more of an indirect warning. Students who do not know what a sequence is have not adequately prepared for the exam and must study.

> Kurt: So a sequence is an ordered collection of stuff where each element is individually addressable or accessible by an integer **index** starting with zero and working its way to the- the end of the sequence. Alright. So strings are sequences, as we talked about last time, strings (.) a string like this: "abcde". it has individual elements, the individual characters, each one's addressable by an **index**.
>
> Audio-Recorded in Lecture 2/22/17

When Kurt speaks of sequences, he nearly always also uses the words "ordered," "collection," and "element."

> Kurt: What's the **index** of the "a"?
> Trina: one.
> Kurt: Who said one? ((point)) Zero works a whole lot better! 'Member we start our **indexes** with zero. We count funny in computer land. Right? So, A is zero, B is one, C is two, and D is three, and E is four. And even though there's five elements, in this sequence, there is no **index** five. **Indexes** go from zero to four for an n-element sequence, the **indexes** go from zero to n minus one. That syntax again is pretty simple.
>
> Audio-Recorded in Lecture 2/22/17

In this example, he uses the term seven times, demonstrating the expected plural form "indexes" three times of the seven. He defines indexes, then quizzes the group, allowing an opportunity for feedback and uptake. Only Trina, a white woman who participates frequently in lecture, volunteers an answer. Her answer is incorrect - indexes begin at zero and the question was about the first character in the string - so Kurt must correct her. First he asks who said it, even pointing in Trina's direction, an act which could have a substantial impact on Trina's positive face. Next, perhaps as a redressive strategy, he indirectly corrects her. "Zero works a whole lot better" is

both a joke and an example of the strategy of positive-valence words. He justifies his correction, "'member we start our indexes with zero," using the abbreviated version of "remember" and inclusive "we" to reduce the perlocutionary force of the utterance, and jokes one more time: "we count funny in computer land." Finally, he elaborates by identifying the index of each element in the string. His final statement, "that syntax again is pretty simple" is likely intended to reassure students that they will not struggle to use indexes in their homework; however, it could have the unintended consequence of damaging the sense of positive face of those students who find the syntax challenging.

Later in the lecture, Kurt repeated that "every one of the items in a sequence is addressable by an integer index, starting at zero," building in the concept of strings and giving more examples. He also demonstrated that while lists can be edited using indexing, strings cannot. This provided an opportunity to define "mutable" and "immutable":

> Kurt: So, remember that strings are immutable. But lists, on the other hand, (.) are sequences that are mutable. They're changeable. Immutable like in mutate, like, you know, (.) zombies are exposed or people are exposed to radiation from some comet and they turn to zombies, the mutate. So think of lists as being zombies. Or not. And again a list is accessible by an inner- integer index, just like a- a- a st- a string is. but, because it's mutable, I can change values, I can grow the list, I can shrink the list, that makes it a very flexible structure that's useful in a lot of ways.
>
> Audio-Recorded in Lecture 2/22/17

Here he uses the common English word "mutate" to anchor the new terms, mutable and immutable. He does not demonstrate that it is impossible to change strings using live coding or group activities, but the TAs do. In both quarters, Kurt later used live coding to demonstrate how indexes can help students in their projects. This involved creating a list and then having the shell display or change different elements in that list.

The introduction of "index," like that of several other class terms, was associated with several other class concepts at differing levels of difficulty. While the professor believed that it should be introduced with lists, some of the TAs introduced it with strings, which was discussed somewhat earlier. As such, the instructors sometimes discussed indexes before students would need to be able to use them and then once more during the lists unit, giving the term a less robust modeling phase than they might have if they had only expected to have one modeling period.

Student uptake of "index" and related words

Students began needing to work with indexes the following week. In the beginning, students who felt confused about what indexes were for would ask very general questions about it, as Katherine, a highly participatory Asian-American student, did at the end of John's section:

> Katherine: I'm confused how to use the **index**. I haven't really reviewed it, so.
> John: So, if you want to use it as the **index-**
> Katherine: Mhm.
> John: Then you need to do it every time on this.
> Katherine: For the-
> John: Not just-
> Katherine: The error.
> John: Yeah
> Katherine: Oh.
>
> Audio-Recorded in John's Section 11/4/16

Because John and Katherine are looking at her computer, they use very few class terms. However, Katherine is prepared to use the term "index" in her problem statement asking for guidance: "I'm confused how to use the index." John types into her computer to demonstrate the proper use of indexes, which Katherine appears to be confident she understands based on her feedback.

Instructors often avoided correcting students' non-expert speech in the interest of rewarding a general understanding of the class content. Just after this exchange, the new

homework assignment is released, which requires students to create a Morse code translator function. The assignment provides a list containing lists that each have an alphabetic character and its Morse code equivalent, and students must write a function that can find the character that is in the next position within the same list as a given Morse code entry. The way to determine whether an element is in a particular position in a list is by using indexes. After it was turned in, Chris demonstrated the solution in section using a list called "python" and a list called "items":

> Chris: On our way we have to make some type of comparison, so I want to say, "if 'ch' is equal to," and then one of our second and we'll see if it's equal to.
> Oleg: Um, the first, uh, **index**, of the, um (.)
> Chris: The first **index** of Python, exactly! Which, uh, you get by item 'z', right? So that's saying "if they're equal, if my character is equal to that item," so now I want to replace it, with this **index** of items, right? Um (.). So we need some type of out character, right? And, what am I going to assign this out character to?
> Oleg: The second.
> Chris: The second, exactly! ((writes on chalkboard)), right? Ok, so now I've got that. Um, there's a couple other things that we have to do, right? 'cause we have to add some extra spaces in, and we have to uh, add in the hashtags, right?
>
> Audio-Recorded in Chris's Section 11/8/16

Here Oleg is prepared to use the word "index," but not necessarily the way a full member of the programming community would. Referring to "the first index" is not a typical formulation: rather, one would say "the first element" or "index zero." Unlike a parent in the case of child language socialization, Chris replicates Oleg's non-expert form, repeating "the first index of Python, exactly!" Likely his desire to encourage Oleg by bolstering his positive face is outweighing a desire to model expert speech. The issue is avoided in Oleg's answer to the follow-up question, due to his use of ellipsis: he says "the second" instead of "the second index." Chris mirrors his syntax here, again repeating his words and adding "exactly!" TAs frequently supported and even repeated non-expert linguistic style when produced by students as long as their answer was factually correct. As in the case of "function," TAs were able to infer that

students were asking about the target term without requiring them to use it, explaining solutions in exact detail, and even taking control of the keyboard and typing for them when their directions were not followed. In the latter case, the TA often followed a similar script pattern to a live-coding event, asking for students to fill in answers and volunteer information as the TA typed the correct code into the student's computer.

Toward the end of the quarter, some students began to show comfort using "index" as a verb and as a preposition, rather than exclusively as a noun. This comfort could constitute feedback in the form of an epistemic stance demonstrating confidence. At the end of spring quarter, Richard had a question for John in section:

> Richard: Uh, I just wanna make sure I'm getting like I guess **indexing** right, in the lists.
> John: Mmhmm.
> Richard: So it says X **index** zero, does that refer to the tuple, the very first one, "a comma b," or is it referring to the first, I guess, entry in the first tuple?
> John: So for X in pairs, right?
> Richard: Uh huh.
> John: So there are actually three elements in this list.
> Richard: Right.
> John: Which is, which are three tuples. Right. So X will be this, uh for, for the first time you run this pro--, uh this, uh for loop, it will be the first tuple, and then it will be the second tuple, right, and the third. And so if the X zero, X, so X will be this tuple, for the first time right, and in this tuple, the first element should be a string A right?
> Richard: Yeah.
> John: So X zero will be A, string A.
> Richard: Shouldn't it be X zer--, or, index zero, and then another brac-- yeah, bracket zero?
> John: Uh, if your X is pair, then that is correct, but over here "X- for X in pairs," right? So X will be each element inside this list's list's lists.
> Richard: Okay.
>
> Audio-Recorded in Section 3/13/17

Richard is asking about a relatively complex concept using expert-style metalinguistic coercion. Richard identifies the topic of his question as "indexing," applying the morpheme -ing similarly to John and Kurt's modeling. He also properly relates the list name "x" and the index number

"zero" using "index" as a preposition: "x index zero." Richard's repair toward the end of this exchange provides some insight into his concept of the correct use of index as a preposition: "Shouldn't it be X zer- or, index zero" suggests that he is dissatisfied with having the list name (X) modify the index integer (zero) without an intervening word. Instructors have modeled "index" and "of" in that space, and Richard chooses to use "index." The use of "entry" in "the first, I guess, entry in the first tuple" is an acceptable but less common way to refer to an element. John continues to use "element" in his answers, providing Richard with new examples of the more preferred word. Once his question is asked, John gives a detailed answer with examples, to which Richard only has to provide backchanneling to provide assurance to John's discourse marker "right?" When he requires it, he asks for clarity: "shouldn't it be…", which is a question phrased as a dissenting opinion. Positing a hypothesis about how the indexing would work could threaten Richard's positive face, suggesting that he feels secure enough to be able to withstand correction from John. When he receives it, he answers, "okay," signaling satisfaction and understanding. This fluent use of "index" and "indexing" is somewhat more common in spring quarter than fall quarter, but examples exist in both sets.

Pair Programming and Student Uptake

In the last pair programming session, students were asked to use a loop to add elements of two lists together. Two women students, Tanya and Brigitte, exemplify uses of "index" as acquired by the students toward the end of the quarter:

> Tanya: So, we have the sum, we just wanna add the sum into the new list.
> Brigitte: Um, so this would be result equals (.)
> Tanya: Equal to (.) new_list?
> Brigitte: Of the sum?
> Tanya: The sum is a- is a integer, not a list.
> Brigitte: Uh huh. Exactly.

Tanya: Can we use the (.) this?
Brigitte: I don't (.) think so?
Tanya: [inaudible]. ((run module)) Yeah, we can!
Brigitte: We can?
Tanya: Yeah.
Brigitte: Kay, 'cause-
Tanya: Yeah, that would be it. It just returns like that.
Brigitte: Mhm.
Tanya: Should I return here, or?
Brigitte: No on that- yeah. Let's see if this works.

Audio-Recorded in John's Section 3/6/17

As is typical of this activity, the students largely disregard the strict rules of the driver/navigator relationship in favor of an approach that favors co-construction of knowledge. Brigitte and Tanya produce utterances of approximately equal length, providing information and suggestions of approximately equal quantity. Both are using inclusive "we" to describe their actions, where a driver/navigator relationship would likely be characterized as one person issuing directives to the other in first-person singular. When Brigitte wants to generate the result, she is unsure which variable it should be set equal to, so she pauses. Tanya supplies the correct answer, "new_list." Brigitte had thought it might be a different variable, "sum," but Tanya reminds her using an unmitigated declarative: "the sum is a- is an integer, not a list." Brigitte quickly accepts her disagreement, boosting this acceptance using repetition: "Uh huh. Exactly." Tanya wonders if it is necessary to create a new variable called "result" when their result is a list they have already created called new_list, but Brigitte is unsure. She signals her lack of confidence using indirect speech (reporting on her thoughts rather than baldly saying "no"), by pausing, and by use of high-rising terminal. Tanya chooses to test the code, employing the expertise of the computer to resolve an area of difficulty for the two novices. When it is successful, they have their answer without the intervention of an instructor. Next, the issue of the index comes up:

Tanya: Wait, I think I should have **index** here. 'Cause it's in the while loop.
Brigitte: Oh yeah.

Audio-Recorded in John's Section 3/6/17

In this case, "index" violates the normal syntactic and semantic features that the word would normally have. Tanya is hoping to convey to Brigitte that the word "index" needs to be typed, which means she is referring to the word itself and not using it. Normally index would be a noun, requiring a determiner (e.g., "an index"). This type of coercion can be considered metalinguistic. In this case, the technical term comes from the object-language, Python, and the sentence in which it is placed is the meta-language, English. Tanya and Brigitte continue to design their solution:

Tanya: So, [inaudible]. Do I need to **index** this one?
Brigitte: Mm, (.) I don't know if [inaudible] should go in the while or in the for.
Tanya: The for. So I should run.
Brigitte: Try it like that and but I think we can change it if it returns an error.
Tanya: Yeah I think I should. This i is [inaudible]. ((run module))
John: Yeah Friday is 93.
Tanya: ((yawns)) So, (.) I think we should just- oh I think I don't need to [inaudible]. Because, um, the length of- the length of the list, like t- the- t- the list t is equal to three,
Brigitte: Mhm,
Tanya: but the **index** can only go to, 'cause it's at zero, one to that,
Brigitte: yeah,
Tanya: so if I- I have at one, the **in- index** can only go to two. Because the length equal to three, the maximum value here is like (.) if this a three can only go like three minus one, you see what I mean?
Brigitte: uh huh, uh- okay so try it like that?
Tanya: It still say [inaudible].
Brigitte: I'm not sure why.
Tanya: Hm.

Audio-Recorded in John's Section 3/6/17

Only Tanya uses the term in this conversation, four times, all in an expert-like style. She omits a determiner before her first utterance of "index," which may be attributable to her status as a non-native speaker of English rather than the degree to which she has taken up the proper use of

"index." Throughout this exercise, whenever a student expresses an epistemic stance indicating lack of certainty about which course of action to take, the solution is typically to run the program and see what happens. In a demonstration of the limitations of a short pair-programming session, the students run out of time before they can find the solution to their problem, leading to this exchange:

> Tanya: But we don't really need to find a solution really. As long as we try ((laugh))
> Brigitte: Yeah.
>
> Audio-Recorded in John's Section 3/6/17

Tanya and Brigitte would have liked to find the answer, but they jokingly acknowledge a discourse about education: teachers would rather their students try hard than get the answer exactly right. The students negotiated meanings and furthered their knowledge of iteration and indexing in their conversation with each other, which they recognize as valuable.

Conclusion

In the process of learning how to speak as expert programmers speak, students listened to the experts at their disposal. As a veteran instructor, Kurt was careful to repeat target terms frequently, modeling them in their various syntactic and morphological permutations. The TAs also demonstrated this practice, albeit less consistently and less frequently. The degree to which Kurt repeated the target terms might sound somewhat absurd to a professional programmer, but in the context of language socialization in particular and education in general, it is a technique that seeks to take advantage of the rhetorical and mnemonic power of repetition. From the perspective of metalinguistic coercion, it is important for modeling the ways in which each term can appropriately be integrated into English grammar.

While uptake was still in process, student feedback took the form of demonstrating several avoidant behaviors: generalized problem statements, pausing in the place where the term might have been appropriate, use of gesture and deixis among them. These behaviors constitute epistemic stances of lack of certainty, which were supported and allowed by the instructors, but less so by peers. When students used terms in non-expert ways, Kurt did not support this use; however, TAs and peers tended to. Asking for guidance from experts appeared to be associated with somewhat more avoidance and fear than asking peers, and conversations with experts tended toward the lopsided, with novices mainly backchanneling and sometimes asking follow-up questions. Additionally, taking the keyboard away from students and typing directly into their assignment files appeared to be a common practice among TAs.

CHAPTER 5: POLITENESS THEORY AND NEGOTIATING MEANING

If learning is the process of co-constructing meaning in the world, instructors are not the only interlocutors who can provide meaning-making opportunities for students; peers can generate such opportunities themselves. The purpose of the pair programming activities was to increase the amount of time students spent engaging in peer socialization, giving me the opportunity to compare the social and grammatical features associated with speaking to instructors with those of speaking to peers. In this chapter, I describe the tendencies of students in both quarters as they speak with instructors and that of students as they talk to peers about course concepts, employing politeness theory as a tool for measuring the amount of threat they perceived to their positive and negative face in these cases.

Instructors with Students

When students and instructors interact, students have an opportunity to hear experts using terms in a manner consistent with full membership in the community of computer science professionals. When students use class terms in the presence of instructors, those instructors can estimate the degree to which their students have mastered the material, and they can choose based on this information whether to adjust their approach. To the extent that the students' success in ECS 10 is contingent on this dynamic, the analysis presented in chapter 4 offers insights into maximizing the positive effects of instructor-student interaction. This section focuses on identifying the extent to which power differences might negatively affect students'

and instructors' ability to freely identify and pursue knowledge gaps in two commonplace situations: asking for and volunteering course-related information.

Asking for Clarification: Positive and Negative Face Threat

When students are unsure of the meaning of a new term or how to use a new tool, they may ask the instructors during lecture or section, they may attend lab, or they may approach them after class. Such question-answer exchanges often include follow-up questions ranging from a request to answer again (in the case that the student did not understand the answer) to a reformulation of the answer (as a bid for validation that they had understood the answer). Asking for clarification may involve a threat to the asker's positive face because they must admit to ignorance concerning a course-relevant concept, a threat to the hearer's positive face because asking for more information may imply that the initial explanation was not sufficient, and a threat to the hearer's negative face because asking for clarification after class or in lab is a bid for the instructor's time and attention.

In lectures and discussion sections, instructors often modeled asking for help, taking on the role of a confused student. This may have been intended as a method to begin socializing students into the practice of asking questions of them. One socialization event that tended to feature this practice was live coding, in which instructors would write a function in their own development environment, which was projected on the board. Sometimes instructors would write the entire function themselves, narrating as they went, and sometimes they would ask students for directions, modeling question-asking at the same time:

> Kurt: For value in X colon, we indent (.) and somebody other- somebody else! How do I add these values into total?
> Felipe: ((raise hand))
> Kurt: Yes!

Felipe: Uh, print total plus value?
Kurt: ((laughs)) Outstanding! Look! The dead have risen! They are coming to life!

Audio-Recorded in Lecture 2/22/17

When he modeled question-asking, Kurt tended to favor first person singular and first person singular pronouns. This is a way to increase solidarity as well as a method for modeling the social practice of asking questions. He also used humor as an immediacy behavior, reducing social distance, improving classroom affect, and encouraging communicative behaviors in the class. He was also careful to ratify the student's answer emphatically. In conversations with the instructors, several of them indicated to me that they hoped students would come to them only with specific questions like this one rather than bringing non-functional code and asking for general help. Students almost never asked questions this way, instead favoring either problem statements (e.g., "I don't know why this isn't working") or requests containing several politeness strategies but not much detail (e.g., "Sorry, could you maybe look at this, please?").

After each lecture and discussion section, a line of students would form at the front of the classroom. These students were planning to ask a question of the instructor. There was no set time after class for such an activity, and the professor stated at the beginning of the course that students with questions about course concepts or struggles with their homework assignments should go to the lab to ask a TA for help. Even so, a large and diverse subset of the student population appeared to consider asking for help after class to be an appropriate activity, as between five and thirty students approached after each lecture. While most questions posed by students after class were about grades and policies, some students asked about course concepts. Students asking for help often employed several face threat mitigation strategies, suggesting that the students perceived substantial threat to instructors' negative face. For example, when Rachel approaches Kurt to ask about data types, she asks her question this way:

Rachel: Hi. Uh, I kind of have a question for the multiple choice part. Uh, like there's a question, uh, like, we can't compare a string to a number in the Boolean expression?
Kurt: That's right.

Audio-Recorded in Lecture 2/22/17

Rachel's question is extremely specific compared to those typically asked by her classmates: she wishes for clarification concerning how Boolean expressions interact with data types. In this excerpt, she is summarizing the facts she knows that led her to her question. By using high-rising terminal, Rachel is indicating that this is only the first part of her utterance, which Kurt recognizes by ratifying her statement but not beginning his answer yet. The prevalence of discourse markers "uh" and "like," indicating ongoing formulation of thoughts, suggest that Rachel might feel flustered or nervous about making sure her statement is formed correctly. Rachel makes use of some face threat redressive strategies here: she mitigates using minimizing language ("kind of") and uses justification for her demand on Kurt's time by declaring "I have a question." Once her introduction of the topic is complete, and Kurt verifies that she has so far understood the issue, she continues:

Rachel: Yeah. And I was just wondering, like, what if we added an int function to, like,- What if we co- uh, like, um, turn the string into a number?

Audio Recorded After Lecture 2/8/17

After accepting Kurt's evaluation of her claim, she asks the body of her question. Again, several instances of "uh," "um," and "like" suggest significant focus on the formulation of this utterance. Here Rachel distances herself from her question using the past continuative verb "was wondering" and more minimizing ("just"). The use of second person plural "we" here may be an attempt to reduce social distance or interactively position Kurt as being on her side. This

question contains two false starts with repairs, creating fragments of three different equally correct ways of phrasing the question: "what if we

(1) added an int function to convert the string

(2) convert the string

(3) turn the string into a number?"

All three would be proper ways to describe the process she is asking about, but she interrupts herself during both of the phrases that contain class terms (int function, convert) and settles on a non-technical verb "turn." Although "number" is not a class term and does not refer to a specific data type (students are supposed to differentiate between integers and floats), Kurt supports and takes up her use of the word as he adds specificity to her question:

Kurt: If you turn the string into a number, if the- if the string is- is actually a number in quotes-
Rachel: Yeah.
Kurt: Then that would be fine.

Audio Recorded After Lecture 2/8/17

Kurt's answer will only make sense if the string is an integer in quotes: applying the integer function would then convert it to an integer, allowing it to be compared to other integers as she asked. Kurt's answer is only five words long: "Then that would be fine." In the end-of-quarter questionnaire, several students wrote that his answers felt curt and sometimes left them feeling silly for having asked. This may be one of those times, although it is important to note that there were several students waiting to speak with him, which likely incentivized him to keep things short. Rachel is not completely satisfied with her understanding of the concept, and perhaps with the simplicity of the answer, so she asks a follow-up question:

Rachel: Oh (.) like, what would- what would the answer be? Like, would it be- the answer be- Like, if the- If N is less than uh, one-twenty-
Kurt: Mhmm.
Rachel: Then the-

Kurt: And then if you typed in a number and then converted it-
Rachel: Uh huh.
Kurt: Then it depends on what the number is.
Rachel: Oh. Okay.
Kurt: Okay? 'Cause then- then- then you go through the if then else stuff.
Rachel: Okay.
Kurt: Okay? So it all depends on that. But yeah. The difference there is that it didn't get converted.
Rachel: Okay.

Audio Recorded After Lecture 2/8/17

Once Rachel's follow-up question is made clear, Kurt begins to answer in much more detail. He continues to speak in simple sentences, suggesting a desire to finish this conversation quickly. Rachel backchannels "okay," indicating acceptance of his answer and perhaps even understanding. Kurt's use of "okay" differs from uses of the discourse marker in other contexts: in lectures, discussions, and labs, it appears at the end of an utterance and functions as a request for validation or affirmation. Here, it is a repetition of Rachel's backchanneling, which could be interpreted as a negative stance on the conversation as a whole. Perhaps this was an expression of Kurt's explicitly stated disapproval of asking questions outside of lab or office hours. It appears to have translated to students both in their tendency to avoid more than one follow-up question and in their answers to the questionnaire, but it did not stop them from continuing to ask questions after lecture.

In lab, students were expected and encouraged to ask questions of the TA. Questions tended to be generalized problem statements or requests for help, rather than specific questions employing class concepts or terms. As they did after classes, students had their code displayed and would gesture at it to indicate their problem rather than describe it. For example, during an early lab session in the fall, Rheanna signaled for Chris to join her at her computer:

Rheanna: ((show screen)) Like, I just- I can't get it. Like-
Chris: Sure. All right. We've got (.) okay. So you actually have everything lined up exactly how you want it to be. We just gotta change a couple things around. Right?
Rheanna: 'Kay.
Chris: So we want this body mass function to do everything that you've written right here, right?
Rheanna: Mhm.
Chris: We want it to take in a name, we want it to get meters, we want it to get kilograms, and then we want it to, uh, print what the body mass index is and figure out those values, right? And we want to use these functions that we've already written up here to be able to do that.
Rheanna: Yeah.
Chris: Right? …

Audio-Recorded in Chris's Lab 10/11/16

Rheanna's question is "I can't get it," a problem statement accompanied by a gesture toward her homework assignment. Chris accepts the perlocutionary force of this problem statement, replying "sure," as though to a request, ratifying her formulation. A problem statement flouts the Gricean Maxim of relevance, leading Chris to interpret the conversational implicature that she is requesting help. He begins describing the issue and its solution in detail, to which Rheanna only has to backchannel. Like many other students, Rheanna asks her question with some hesitation and markers that indicate trepidation or lack of confidence, avoiding class terms or concepts. While asking Chris to fix her code feels threatening to Chris's negative face, he moves to reassure her that this is not the case, acquiescing with repeated discourse markers indicating his willingness to help: "Sure. All right." He also seeks to reduce the damage to Rheanna's positive face now that she has admitted to not knowing what to do. His use of the first person plural "we" serves this function, and his initial positive assessment of Rheanna's function design: she has everything organized "exactly how you want it to be." Chris uses mitigation when referring to the steps she will need to take to fix the problem ("just") and refers frequently to the code that already exists to remind her that she has done good work. As discussed in Chapter 4, Chris and

the other TAs spoke overwhelmingly more words than the students in these interactions, and they often took control of the mouse and even the keyboard, typing directly into the students' assignments as they explained how to solve the errors they saw.

When students wished to ask questions of their instructors, these questions were characterized by several face threat redressive strategies. Common strategies for repairing the negative face threat to instructors included minimizing, indirectness, justification, and point of view manipulation (use of "we", for example). These strategies, particularly the last three, were much more common in questions asked after lecture or discussion section, as these events were not officially encouraged by instructors. Students' avoidance of multiple follow-up questions in this context can be explained the same way. Students' frequent use of um/uh/like, pauses, and repairs suggest students perceived a strong threat to their own positive face. In lab, instructors took steps to preserve that face by reacting with enthusiasm to requests, long and syntactically complex answers, and minimizing their acknowledgment of the degree of the students' knowledge gap. After class, this was not the case, perhaps as a response to the increased negative face threat perceived by the instructors.

Answering Questions: Positive Face Threat.

Students are often asked to supply information that is already known to the instructors. The most common example of this type of event involves the instructor reviewing material in front of the class and asking students to fill in class content. Due to the difference in expert status between the students and instructor, it is assumed that the instructor already knows the answer and is assessing the students' answers according to their own expert knowledge of the ideal

answer. This changes the perlocutionary force of a question asked by an instructor from a request for information to an assessment of the student's competence.

The conditions under which instructors tended to ask students to produce course content were overwhelmingly in group settings, either during lecture or during section. Often, this took place during a live coding event. The instructor would declare that the class would be writing a function together, would describe the requirements of the function, and would ask students to volunteer crucial pieces of that function. They did not single people out, only waiting for someone to raise their hand or speak up. For example, while live coding a function that adds up all the integers from one to some user-supplied integer, Kurt asked students to help design it:

> Kurt: So, we call that, that last number N, so let's just call it N here. What kinda things do I need to keep track of inside of this function?
> Jack: What N is.
> Kurt: What N is, okay well, that sorta passed through, what other things do I need to ca--, uh, keep track of?
>
> Audio-Recorded in Lecture 10/21/16

Having written a function definition with a variable called "n" as its parameter, Kurt asks the class what variables need to be defined in the body of that function. To do this, he asks, "what kinda things do i need to keep track of inside this function?" By asking in such a general way without using the term "variable," or "define," Kurt is signaling to the students that he expects them to be comfortable already with this portion of the functional design process. Jack supplies an answer that takes the form of a sentence fragment "what n is" with a high-rising terminal, which may indicate a guess or a lack of confidence in the answer. Students frequently used high-rising terminal or question words ("wouldn't it be" or "is it") when supplying answers, but they also often answered using sentence fragments containing no features of a question. His answer is incorrect in that n is already passed as a parameter, although he is right that n is important and will be used later in the function. Regardless of whether students' answers are correct or

incorrect, Kurt always repeated the answer before responding to it. After repeating "what n is," Kurt says "okay, well, that's sorta passed through." The discourse marker "okay" here signal that the following statement will be a departure from the previous one, while also managing Jack's positive face by recruiting the connotations of agreement even as he does not ratify the answer. "Well, that's sorta passed through" means that the variable n is already part of the function because it was passed as a parameter: by listing it in the parameters of the function definition, it is already handled. This is a justification for Kurt's rejection of Jack's answer, which also contains mitigation ("sorta"). He re-asks the question:

> Kurt: what other things do I need to ca--, uh, keep track of?
> Michael: The result.
> Kurt: The result. Total equals, what should I start my total at?
>
> Audio-Recorded in Lecture 10/21/16

By adding the word "other" in his repeated question, Kurt reinforces that Jack is not wrong about n needing to be tracked. "Other" implies that a satisfactory example has been provided and that Kurt now wants to see more of them. Coupled with his justification for not quite accepting the answer Jack gave, this question implies that while n will continue to be used in the function, it is not what Kurt was looking for. Michael answers with another fragment, "the result," delivered with no high-rising terminal. Kurt repeats, "the result," with no discourse markers as indications of his stance, but makes clear that he accepts the answer by asking a follow-up question. Rather than ask students to invent a name for the variable that will hold the result, Kurt names it himself. Choosing a name for a variable is not a target skill at this point in the lecture. Students have done this several times and already know the rules for how to choose appropriate names. Kurt asks instead for the value that "total" should be initialized to. This is the number that the integers will be added to to give the sum, so the correct answer would be zero:

Kurt: Total equals, what should I start my total at? I'm gonna add all these integers at, what should I start this thing in?
Michael: One.
Kurt: Really? I'm gonna add a series of numbers together, you're gonna keep a running total in some variable. Where do I wanna start that variable value at? Zero. 'Kay. Alright.

Audio-Recorded in Lecture 10/21/16

Michael here gives an unequivocally incorrect answer. If the sum started at one, the result would always be too high by one. He delivers this answer as a fragment again, in a downward intonation indicating a declaration. Kurt's answer, "Really?" is a performance of surprise, an emphatic non-ratification of Michael's answer. He reiterates the question in more detail, indicating that he expects Michael to change his answer now that he has been reminded of the role "total" will play. While this is never explicitly summarized for the students, they are expected to know that when adding or subtracting, the total should be initialized to zero and when multiplying or dividing, the total should be initialized to one. This is considered part of the "basic arithmetic" (lecture, 9/21/16) they need to know before taking this course. Rather than accept answers from the class when he re-asks the question, he answers it himself. This may be because, based on the "basic arithmetic" summarized in this paragraph, he believes this question to have only two possible answers: zero or one. One has been revealed to be the incorrect answer, so zero is the only available option. This interpretation is reasonable because of the understanding he and the instructors have about how the "total" variable should work; however, it may be that Kurt has entered a section of the live-coding in which he does not wish to reach out to students for answers, as he answers the next question himself as well:

Kurt: And now I'm gonna add up a bunch o' numbers from one to N. So I can start with N and work my way down, or I can start at one and work my way up, I'm gonna choose to start at one and work my way up. (.) With me so far?

Audio-Recorded in Lecture 10/21/16

Rather than ask the class whether they should start with n and work down or start at one and work up, Kurt chooses for them. This is likely because the students are just now learning how to write while loops for the first time, so Kurt would prefer to model their design than ask the students to supply the design, an activity that would be more appropriate after the initial modeling phase. If this is the reason for his choice not to ask the students to choose whether to work up or down, the explanation hat Kurt supplied the answer "zero" earlier because he is generally disinclined to ask for student answers in this phase of live-coding is not necessarily appropriate: defining "total" as equal to zero is not part of the new skill set being developed, so it is not part of this pause in asking for student responses due to new material. Students do know the basic anatomy of a while loop, however, so Kurt returns to asking students to supply the contents:

> Kurt: And then, the time for a loop. While, while what?
> Daryl: Counter is less than or equal to N.
> Kurt: Counter, less than or equal to N. Excellent.
>
> Audio-Recorded in Lecture 10/21/16

Daryl has answered correctly, so Kurt repeats his answer exactly and then finishes by indicating approval: "Excellent." Because the answer is correct, Kurt has no additional comment. He moves on. As he continues, students continue to supply correct answers in the form of sentence fragments with downward intonation. Kurt accepts those answers by repeating them either with no comment or with a marker such as "okay" and "very good." All the answers were supplied by men except the last one:

> Kurt: And then what?
> Rachel: Return total.
> Kurt: Return total!
>
> Audio-Recorded in Lecture 10/21/16

This answer was accepted by way of an exact repetition, as usual, but with an emphatic tone: it was spoken with higher volume and higher pitch than the rest of Kurt's speech. Over the course of both quarters, men were much more likely to supply answers in group contexts than were women. For the most part, this was because they volunteered information more often and more loudly, without waiting to be called on. In some cases, both men and women would raise their hands to answer, and in this case instructors nearly always called on men.

Throughout both quarters, student answers to instructor questions tend to have almost no positive face threat redressive strategies associated with them because they are in no danger of damaging their instructor's positive face: it is the students who are being evaluated. Instructors have several strategies for preserving their students' positive face including justification, attempts to locate some aspect of the incorrect answer that could be correct, and discourse markers that imply agreement (e.g., "okay") to introduce utterances containing disagreement. Correct answers were met with an interjection or emphatic repetition. Students and instructors do not appear to detect a need for negative face threat redressive strategies. Students do not need them because they are being asked a question, so they are responding to an instructor's request for them to speak rather than initiating conversation. Instructors do not need them because they are aware of the power relations between instructors and students: instructors have the right to ask students to supply course information on demand: this is considered a natural feature of the student-instructor relationship.

Pair Programming Activities

In pair programming, students interacted almost exclusively with a peer. Only rarely did students signal for an instructor to approach and clarify a point of confusion for them. Speaking

to a peer generally entails much less threat to both positive and negative threat. The speaker is not concerned that someone who has control over their grades and a higher level of expertise will hear them misspeak, the speaker is not worried that they are impinging on time the hearer would prefer to spend doing something else, and neither party is in a position to damage the self-image of the other when answering questions.

Asking for Clarification: Positive and Negative Face Threat.

During pair programming activities, students were asked to complete programming problem sets together. This assignment was created with the intention of prompting moments in which one student needed a course content-related question answered. Most of the time, when students in the pair programming context did not know something, their partner did not know it either. This meant that the pattern of asking for clarification largely took this form: (1) Student 1 indicates that they have a knowledge gap, (2) Student 2 gives non-answer, (3) Student 1 or Student 2 runs module to see what happens, (4) Python returns a successful or unsuccessful result, and (5) Students indicate that clarification was achieved. This contrasts significantly with the process for students asking instructors: (1) Student asks for clarification (2) Instructor explains the concept or demonstrates relevant code, and (3) Student indicates that clarification was achieved. The second scenario involves fewer conversational turns, but in the following section I will demonstrate some of the benefits of asking for clarification from someone who cannot provide it themselves.

In the following example, Victoria and Brittany are unsure what will happen when they have a string (a single period '.') that is multiplied by an integer. When a string is multiplied, it is printed the number of times equal to the integer it is multiplied by. For example, the string 'abc'

multiplied by three will return 'abcabcabc.' When these students assign the string '.' to a variable called 'delimiter' then multiply that variable by five, the result will be '.....' As the first step, Brittany proposes the existence of a knowledge gap:

> Victoria: And then we do "delimiter times five."
> Brittany: And that should be- I don't kn- no idea what to expect from this ((laugh))
>
> Audio-Recorded in Chris's Section 1/25/17

Brittany's utterance "I don't kn- no idea what to expect from this" can be considered a problem statement: she is declaring the existence of a situation that could hinder their completion of the task. She begins "I don't know," a simple subject-verb construction with no politeness strategies associated with it, but repairs to "no idea," a stronger phrasing. By boosting, Brittany is drawing attention to her lack of knowledge, and her laughter at the end is further evidence that she has little fear of losing positive face by admitting ignorance or impinging on Victoria's negative face by declaring the need for clarification. Victoria performs step two: a non-answer.

> Victoria: Me neither.
>
> Audio-Recorded in Chris's Section 1/25/17

In some cases, the non-answer is a simple and explicit agreement that the second student does not know either, in some it is silence, and in others, the student suggests running the module to find out. In this case, Victoria gives a simple, single pragmatic marker signaling agreement with Brittany's feeling of ignorance. The third step, running the module, is undertaken by Victoria, at which point Python returns the string '.....' (step four), and she and Brittany both move into step five, which takes some negotiation:

> Victoria: ((run module)) Oh, multiplying a number times a string- (.)
> Brittany: Oh. Got it. Got it. ((laugh))
> Victoria: Wait, can you explain why it did that, 'cause like I'm confused, like I understand that it's a string, right?
>
> Audio-Recorded in Chris's Section 1/25/17

Victoria sees the output, five periods in a row, and begins to indicate understanding with the interjection "oh," but as she begins to explain it she recognizes her own lack of understanding. The sentence begins, "multiplying a number times a string," and then cuts off as she realizes that she is not able to finish it. Brittany now understands what has happened, so she repeats the interjection "oh," then says "got it" twice, using repetition to boost her expression of joy. She punctuates this with laughter. Victoria employs the marker "wait," a request to not move forward with the assignment. She needs to stop the process as clarification has not yet been achieved for her. Having taken the conversational floor, Victoria asks, "Can you explain why it did that?" A common politeness move in making requests is to ask after their ability. By not inflecting the verb as a conditional or adding politeness markers such as "please," Victoria is indicating that she does not feel a strong need to mitigate the threat to Brittany's negative face. By justifying the question with "'cause like I'm confused," she does acknowledge that in the absence of her need, Brittany would be able to move on. Victoria elaborates on the source of her confusion: she is aware that the object being operated on is a string. She concludes this statement with "right?" a tag question intended to request validation. Brittany explains what she has now come to understand:

> Brittany: 'Cause in delimiter we had this period here?
> Victoria: Mhm.
> Brittany: And it was just, like (.) something. So when we multiply our delimer- delimiter by five, it multiplied what- (.) ever was in the string five times.
> Victoria: Okay, that makes sense.
> Brittany: I think- I think that's what it means.
> Victoria: Okay, that ma- that makes sense. Yeah.
>
> Audio-Recorded in Chris's Section 1/25/17

First, Brittany checks Victoria's foundational understanding of the situation. Victoria has asked why Python returned five periods, so Brittany's answer takes the format of a "because" statement. She directs Victoria's attention to the variable "delimiter" and the statement assigning

a string '.' to it. Victoria backchannels, "mhm," indicating that Brittany should continue. Brittany is unsure how to convey that it is unimportant what is in the string, just that some characters are there. She pauses in the middle of her declarative, "and it was just, like (.) something," but Victoria does not interrupt, so she continues again. Her pauses and repairs suggest that she is not completely certain before she begins that she will be able to provide an expert-style explanation, but she continues. As a continuation of the idea that the contents of a string are unimportant to the function of the multiplication, she uses "whatever was in the string" instead of being more specific about the period. This was an opportunity to generalize the rule about multiplying strings instead of remaining restricted to this one homework problem. The use of "whatever" here appears to be the focus of significant mental exertion, as she pauses in the middle of it just as she did before saying "something." Brittany is constructing knowledge live in front of Victoria, who is checking her proposition against her own understanding. After hearing this explanation, Victoria replies, "Okay, that makes sense," taking a stance of agreement. Brittany positions herself as a non-expert, saying "I think- I think that's what it means," mitigating her string of declaratives by inflecting it through her own perspective. In this case "i think" is not a discourse marker so much as a reduction in the force of her commitment to the truth of her statements. Victoria's repetition of "Okay, that makes sense" is at once a finalization of step five (both parties indicate clarification has been achieved) and a boosting of her support of Brittany's interpretation. Finally, she says, "Yeah," which, when it appears at the end of a sentence, is aa marker that indicates finality, with a connotation that both students are in agreement.

When students needed to ask questions of other students during the pair programming activity, they chose not to mitigate their requests for clarification. Questions were articulated in conjunction with boosting, repetition, laughter, and sometimes justification. Requests tended to

be performed using bald, on-record strategies. The hearer responded with agreement, laughter, and in the event that they knew the answer, syntactically complex answers with frequent checks for validation from the asker. Conversations between students tended to consist of fast conversational turns of approximately equal length rather than long explanations from one punctuated by brief, backchanneling responses from the other.

Answering Questions: Positive Face Threat

While answering questions asked by an instructor may entail substantial threat to students' sense of positive self-worth, the same speech act in the presence of a peer feels less dangerous. When a student answers a question asked by an instructor, it is assumed that the instructor already knows the answer. When a student answers a question asked by a peer, this assumption is not necessarily present. The co-construction of knowledge may take more time to achieve when there is no expert present, but the value of the peer negotiation can be measured in several ways. In the following example from an early pair-programming activity, Taj and Brigitte are discussing how to check whether an integer is odd or even. To do this, they must recall their instructors' description of the arithmetic operator "modulo," sometimes referred to as "mod," which returns the remainder of a division. They will need to test whether the remainder of a number divided by two is zero: if it is, the number is even. If it is not, the number is odd. The modulo operator is represented in Python by a percent sign (%), but instructors always called it modulo or mod, never "percentage." However, the students tended to refer to it as "percentage" for a period of time before they took up referring to it by its proper name:

> Brigitte: Okay. Now it returns true if it's even, so,
> Taj: Then 1 to 10 integer, yeah. And returns true if it's even. Okay, so if, um, number, um, yeah if number um, y- uh, if number that- **percentage sign** two is equal to zero, okay wait- if number, uh **percentage**, because that- if it's divisible by two.

Brigitte: Mhm.
Taj: So like **percentage** of, yeah. **Percentage** two. Not two **percentage**,
Brigitte: 'Cause it would be like-
Taj: Yeah, that's even, right, if it's yeah a n-
Brigitte: But **percentage** gives you like the rem-
Taj: Remainder.
Brigitte: Remainder. So it remains 2. So- like, if you divide a number-
Taj: Yes, I- but if you divide by 2 then, it should be even, right?
Brigitte: So you want a remainder of 2?
Taj: No. So what I'm trying to say is like if number, like **mod** to 3-
Brigitte: Oh, and that equals zero.
Taj: It- equal equal zero, yeah.
Brigitte: Oh okay okay okay ((laughs)).

Audio-Recorded in Chris's Section 1/27/17

At first, Taj uses the novice term "percentage" to describe the symbol he wants Brigitte to use. Brigitte supports this usage, repeating it soon after. As they debate the proper use of the modulo, they negotiate what it means in terms of solving the problem at hand. Close to the end of the discussion, Taj uses "mod," which Brigitte ratifies. While she does not repeat the word immediately after Taj says it, she does use it later in the conversation. The shift from "percentage" to "mod" is extremely subtle and passes without comment from either participant, but from this point forward both participants only refer to it as such. If interpreted as a correction or contradiction, this would be considered off-record, indicative of extreme concern for face threat; however, there is not enough evidence in these cases to suggest that either student is more expert than the other in the use of the term "mod." Instead, this kind of extremely subtle, *mutual* shift may be indicative that the illocutionary force of Taj's use of "mod" is not a correction, and as such "off-record" does not apply. Instead, in the larger statement (deconstructed in the following paragraphs), Taj's illocutionary force is a correction of Brigitte's claims about the use of the symbol (%) in conjunction with its following integer. In correcting this misapprehension, Taj employs several on-record politeness strategies, suggesting little fear of damaging Brigitte's positive face.

The conversation between Taj and Brigitte is characteristic of peer-peer dyads. Where students speaking to instructors demonstrated avoidance, multiple face-threat redressive strategies, and long pauses, students spoke to each other in fast, tight conversational turns and few instances of indirectness. They tended to use terms more freely, despite sometimes misusing them, and gradually moved toward more expert styles together. The speech between students also contained more expressions of joy than that between students and instructors. To demonstrate these examples, their conversation is reproduced here, in parts:

> Brigitte: Okay. Now it returns true if it's even, so,
> Taj: Then 1 to 10 integer, yeah. And returns true if it's even. Okay, so if, um, number, um, yeah if number um, y- uh, if number that- **percentage sign** two is equal to zero, okay wait- if number, uh **percentage**, because that- if it's divisible by two.
> Brigitte: Mhm.
> Taj: So like **percentage** of, yeah. **Percentage** two. Not two **percentage**,
> Brigitte: 'Cause it would be like-
> Taj: Yeah, that's even, right, if it's yeah a n-
>
> Audio-Recorded in Chris's Section 1/27/17

Taj, as the navigator of the activity, is asking Brigitte to type "if number % 2 == 0" into the function body. Brigitte mishears and writes "if 2 % number." To correct her, Taj says "Okay, wait-." The discourse marker "okay" here serves as a part-acknowledgment of Brigitte's contribution as well as a bid for the conversational floor. While Taj has been speaking already, it is reasonable for him to re-bid for the floor because Brigitte's typing can be understood as taking a turn. Saying "wait" is a discourse marker suggesting the non-totality of his acknowledgment of Brigitte's contribution. It both focuses Brigitte on Taj's next utterance and cues his disagreement with her actions. He repeats, "if number, uh percentage," and adds a justification "because that- if it's divisible by two," to which Brigitte backchannels "mhm" without adjusting her code. Taj uses another floor-claiming discourse marker "so," followed by a focuser "like," again trying to direct Brigitte's focus to the difference between his words and what she typed. Taj repeats

"percentage of-," at which point Brigitte moves her cursor to the offending area, prompting Taj to give an interjection "yeah," a pragmatic marker that signals an emotion or stance. Taj then gives an extremely bald on-record correction: "percentage two. Not two percentage." He employs no indirectness other than eliding a verb. Brigitte makes the change and begins to supply her own justification: "'cause it would be like-," although Taj interrupts, finishing her sentence for her. In "yeah, that's even, right?" he gives agreement once more and supplies information, ending with a bid for assurance, the tag question "right?" The two continue:

> Brigitte: But **percentage** gives you like the rem-
> Taj: Remainder.
> Brigitte: remainder, so it remains two. So- like, if you divide a number-
> Taj: Yes, I- but if you divide by two then, it should be even, right?
> Brigitte: So you want a remainder of two?
> Taj: No. So what I'm trying to say is like if number, like **mod** to two-
> Brigitte: Oh, and that equals zero.
> Taj: It- equal equal zero, yeah.
> Brigitte: Oh okay okay okay ((laughs)).
>
> Audio-Recorded in Chris's Section 1/27/17

Brigitte actually does not yet understand that "if number % 2 == 0" means "if a number divided by two has a remainder of zero," which she makes apparent by the next statement. By beginning a declarative with the discourse marker "but," she is taking an oppositional stance. Taj now knows that she disagrees that the code should say what he has told her to write. By fully articulating her perception of the issue, she and Taj are able to find the point of misunderstanding over the next five conversational turns. Brigitte's full declarative statement (interrupted once by Taj supplying a word she stumbles over): "But percentage gives you like the rem- remainder, so it remains two." is less a declarative and more of a request for clarification. She appears to be thinking that the numeral 2 represents the remainder of the operation, not the divisor. Taj supports the first part of her utterance, that the modulo operator returns a remainder, but not that the remainder is two. He begins with "Yes, I- but" to signal partial agreement, then gives the

declarative followed by another discourse marker "right?" Brigitte is still conflating divisor and remainder, so she asks "So you want a remainder of two?" Here is where Taj fully comprehends her misunderstanding. He gives an extremely bald "no," and then uses the class term "mod" to clarify. Once he does this, Brigitte exclaims, "oh," an emotive marker signaling understanding and agreement, and the two proceed to negotiating the other half of the statement. Once the two have negotiated this line of code, Brigitte repeats "okay" three times, laughing as she does so: this expression of happiness at the construction of knowledge that has just taken place is extremely common in the pair programming activities.

A student delivering information to another student tended to take the form of a peer-to-peer conversation in that it was characterized by fast, relatively equal conversational turns with few brief exceptions where the student answering the question needed to explain a concept in detail. Students constructing a schema for terminology use largely did so using unconscious shifts toward correct usage rather than overt correction. Students delivering information about course concepts or processes tended to use bald on-record strategies or on-record politeness to do so rather than highly mitigated or indirect ones. Laughter was common in these events. Students hearing information from a peer tended to reformulate the concept once they felt they understood it, avoiding word-for-word repetition and seeking validation from their partners.

Conclusion

A similar speech act takes on significantly different characteristics depending on whether the interlocutors are peers or expert-novice dyads. In the case of asking for information from a hearer, there are several differences. Where there are repairs, pauses, and filler markers between students and instructors, between peers there is laughter and swift conversational turns. Where

there is mitigation and minimization between students and instructors, between peers there is boosting. Where in the first case there is indirectness, in the second there are bald on-record strategies. Where students asking questions after class avoided follow-up questions, between peers several conversational turns were devoted to reformulating and validating the asker's understanding of the answer. What was most similar was the response of the hearer. Instructors in lab and students in pairs tended to convey enthused acquiescence to the request and gave syntactically complex responses with several attempts at re-wording and checks for validation from the asker. Instructors also took steps to repair students' positive face when they asked questions, directing their attention to the things they had right and minimizing the steps they would need to take to fix their mistakes. This was much less common among peers, perhaps because there had been less threat to their positive face in the first place.

In cases where students were asked to supply course content, the status of their interlocutor was also significant. When instructors asked students to volunteer to answer questions in front of the class, women tended not to answer at all. Students who did choose to answer did so in extremely short sentence fragments, and they often made use of high-rising terminal or question words ("wouldn't it be" or "is it"). Students answering questions posed by peers engaged in more adaptive helpseeking. They spoke in complete sentences, using both on-record politeness and bald on-record strategies to do so, even when calling attention to a knowledge gap in their peers. Conversational turns between students and instructors tended to be highly uneven, with students supplying very short answers and instructors providing long and complex utterances before and after. Between peers, turns were faster and more equal in length. Instructors, upon hearing student responses, would repeat their answers exactly, then provide commentary as to their stance on the answer. Students hearing a peer's answer would instead

reword the answer to test his or her own understanding, seeking validation from their peer. Events in which students spoke to each other demonstrated significant co-construction based on the free nature of the feedback they provided each other as well as the ways in which they came to conclusions (asking each other, testing ideas in the shell, debating). Talk between peers seems to be accompanied by significantly less positive and negative face threat than that between students and instructors across several contexts within the ECS 10 course, and pair programming appears to require more active construction of knowledge from students than do activities involving speaking to an instructor. This increased demand on each student's ability to formulate arguments for or against certain courses of action caused them to be more likely to find themselves in the zone of proximal development.

CHAPTER 6: SOCIALIZING PROGRAMMERS

Language socialization is not only socialization to use language; it is also the use of language to socialize others. At the same time the students of ECS 10 were learning how to use the terminology of their field, they were also learning the discourses that construct the cultural concept of *programmer.* In this chapter, I use social positioning theory and discourse analysis to describe the ways in which the instructors constructed an image of programmers as insular and unwelcoming, nerdy and uncool, and established their expectation that those who would fit in well would be efficient workers. I also describe the politeness techniques they used to ensure proper reception of those expectations. I then explore the degree to which these discourses were taken up by the students, and the related tools they recruited in order to reproduce or resist those discourses. Finally, I use statistical analysis to examine to what extent the introduction of required pair programming could be related to differences in student achievement and retention.

Who Are Programmers?

Over the course of the two quarters in which I observed the instructors of ECS 10, I noted several themes. Thematic coding allowed me to track patterns in the discourses describing the cultural concept of programmers. Speech acts such as giving advice, giving permission, and joking were common mechanisms by which the instructors delivered their beliefs about what is normal and right within the programmer culture. Politeness theory provides an appropriate analytical tool for describing how these beliefs were formulated so as to ensure they were received well. In the speech of the instructors, two themes established themselves: nerdiness (almost exclusively in Kurt's transcripts) and efficiency (in all the instructors' transcripts).

Efficiency was considered a necessary feature of a competent programmer. It could be used to refer to a method for writing well-structured programs or as a description of a program that ran quickly and without using too much memory. Efficiency was modeled by the instructors, was cited as a reason to change something in students' programs, and was referred to explicitly in discussions of good style. Each instructor had slightly different ideas about what efficiency was; Kurt saw it as a way to combat the natural inefficiency of computers, John as a charming type of laziness, and Chris and Anshika as a desirable end in and of itself. All four instructors treated it as an important part of being a legitimate programmer and often constructed it as a positive way for students to engage with their code.

As I examined the nerdiness code, I recognized two related but distinct subcategories. On one hand, programmers were often positioned as highly insular, fierce guardians of the right to membership in their community. Those who failed to meet the standards of membership would be shamed via mockery. On the other hand, they were also socially awkward, lonely, and sedentary, unlikely to have many friends or find love. The first discourse I will refer to as "insular nerds" -- these are the gatekeepers, who are to be feared for their ability to deny legitimate membership. The second I will refer to as "nerdy nerds" -- a more lovable, even pitiable, identity. Of course, these are two sides of the same coin: the same nerd who is too busy or awkward to go out on a weekend is also the nerd who will mock a new programmer for creating a variable name that begins with an uppercase letter. The primary difference between them in the case of this socialization process is that the insular nerds lurk outside the classroom, while the nerdy nerds are both within and without. This construction of programmer identities supports professional discourses that involve ridicule.

Efficient

While Kurt used differential negative positioning to frame programmers as unfriendly, gatekeeping nerds, the TAs did not participate in this behavior. One value all the instructors endorsed, in their own particular ways, was efficiency. Kurt did this using an initial bald on-record directives followed by positive and negative politeness, Chris using justification, and John using jokes at the expense of programmers, and Anshika using modeling, delivered with little to no comment..

Kurt began each quarter by associating programming with lack of efficiency. He described the ability for human language to derive meaning from cultural and historical context, allowing for ambiguity that the hearer resolves by applying that context to perceive the intended meaning. He warned that computers do not possess this ability, and that they must be told all relevant information in order to execute a task:

> Kurt: So what you have to do with a computer program, is you have to be that precise. You literally have to say things like, "Get up off the couch. Walk the ten steps. Turn the door this way." Not in so many words, but it's that level of precision. 'Kay? Everything that you're doing in a programming language is specifying essentially a- a machine, a virtual machine, that performs some duty that- that is a- that- turns into a process when it's executed. So it's very very rigid, it's very very specific, it's not at all ambiguous, whereas human languages are very ambiguous, for efficiency purposes. 'Kay? Computer programming is not an efficient process. 'Kay? It requires detail and precision and attention on your part that you are not used to.
>
> Audio-recorded in lecture, 1/11/17

When explaining the inherent inefficiencies of programming, Kurt employs bald on-record strategies to iconize computer languages and language about computers as highly logical, such as the second-person singular modal “you have to,” which increases the perceived weightiness of the speech act. He uses boosting in phrases like “everything that you’re doing,” “very very rigid, it’s very very specific,” or “not at all.” He also employs repetition, for example,

"not at all ambiguous … very ambiguous." This explanation of the difference between human language and computer languages served as a justification for the warning act being performed here. Kurt refers to the use of language referring to how programs are written and executed as a way of supporting corporate discourses of programmers as empirical and deductive.

Kurt's other mentions of "efficiency" suggest that the inefficiency of computers meant that programmers must therefore be as efficient as possible in order to minimize the pain of having to write such specific commands. For Kurt, efficiency was both the means and end of a better product. Being an efficient programmer meant creating code that would execute more quickly and more simply, but it also meant making the programming process more quick and simple. Techniques for minimizing keystrokes included copying and pasting, abbreviated operations, and keyboard shortcuts. He often encouraged efficiency through modeling:

> Kurt: We could certainly just, you know, have, uh, one variable called, you know, twos, and another called threes, and another called 4s, and that sort of thing. But a sort of more efficient way of doing that would be to take advantage of the data structure we were just learning about, which is a list.
>
> Audio-recorded in lecture, 11/4/16

These instructions tend to be accompanied by point-of-view operations, mitigation, and indirectness. The act of modeling was overwhelmingly characterized by the use of the first-person plural pronoun "we," a point-of-view operation in the form of personal-centre switch that reduces both the positive and negative face threat of advice, which is an established method for instructors to reduce face threat. The positive face threat associated with telling the class how to write a function stems from the presupposition that they do not already know. Using inclusive "we" allows Kurt to counteract this presupposition by putting himself, the expert, in the subject position. The negative face threat is the expectation that students will change their behavior in accordance with his command. Inclusive "we" allows him to mitigate that threat by asserting that

he, too, will follow this advice. Mitigations such as "just" and "sort of" reduce the weightiness of the advice, by reducing the apparent strain on the hearers of the proposed action. Indirect verb forms such as the conditional "a sort of more efficient way of doing that **would be**…" reduce the perlocutionary force of the advice as well. Using these politeness strategies, Kurt suggests that efficiency, a general reduction of keystrokes and lines of code, is a simple and beneficial practice for his students, and will be one that they will share with legitimate programmers, for whom efficiency is iconic.

In addition to employing similar tactics to those Kurt used, Chris was also concerned with giving permission to reduce keystrokes. While Kurt positioned efficiency as an expert practice that students needed to learn, Chris suggested that the students already participated in some of the practices that constitute efficiency, and they needed only to have those practices endorsed by an expert:

> Chris: So let's start our num2 variable. We actually want to do just exactly the same thing as we did before. So we'll just copy and paste, which is totally fine. Don't feel like you have to type every single thing out every single time you have to do it.
>
> Audio-Recorded in Section, 9/27/16

Chris appears concerned that because efficiency is achieved by doing less, students would feel it was not an acceptable practice in a classroom setting. While he used point-of-view operations ("**we'll** just copy and paste") and mitigation ("we'll **just** copy and paste") in his advice, he moved to second person singular ("Don't feel like **you** have to") and boosting ("**totally** fine" and "**every single** time") while giving permission. As he gave the students permission to copy and paste rather than retype, he also employed a direct imperative construction ("Don't feel like"). Because permission entails the hearer's desire to do the action, it is characterized by bald on-record strategies due to its low weight.

Anshika tended to use modeling as a strategy for conveying the desirability of efficiency. She was by far the least frequent participant in this discourse, and those events in which she did take a positive stance on efficiency were subtle. For example, while live coding a function counting the number of vowels in a string of characters, Anshika asked students to provide the simplest method:

> Anshika: Okay so (.) what is one way I can go through each character in the string using a for loop[3]? (.) How do I go through each character in the string? So this sentence that we'll pass as an argument here would be the string. (.) String iteration with for loop, the simplest method. (.)
> Robert: "For i in the sentence."
> Anshika: Okay. "For i in sentence." So the sentence is a sequence of characters, so "for i in sentence" means "through each character in the sentence."
>
> Audio-recorded in section, 11/4/16

Anshika, like the other TAs, was enrolled in a course about teaching computer science. This course consisted mainly of classroom management techniques and logistics. Anshika told me early on in the quarter that she had been instructed in that course to pause after asking questions of the class until someone answered, even if it took a very long time. One way she coped with the discomfort of long pauses was to rephrase the question, as she did in the above example. While she asked for the simplest way to iterate over the string with a for loop, "for i in sentence" was actually the only way to begin such a thing, so what she achieved was eliciting the only option rather than the simplest of several choices. It is possible that she had hoped to see a student design the entire loop, but the answer she received at least allowed her to move forward. Rather than provide an entire solution, Robert instead gave the required language for beginning a for loop, which Anshika took up and elaborated. She went on to provide the rest of the function:

[3] A for loop is a construction that allows a program to repeat a block of code a particular number of times. The programmer can specify the number of times to repeat that code by specifying a range of numbers or a list for the loop to iterate over.

Anshika: So if a particular character is an A (.), then I will increase that by 1. (.) So similarly if the particular character is an E (.) so I could keep doing this with a lot of if statements, five if statements. Where- just to make it (.) shorter, I can just combine them all using (.) ors here. (.) So the- if character is A, or if it's E, (.) or if it's I (.) or O. (.) The character is A or E or I or O or U. Then I could count.

Audio-recorded in section, 11/4/16

Anshika first describes an inefficient solution, a sequence of if statements, each checking whether the character in question was one particular letter. Instead, she suggests the class "make it shorter" - shortness and simplicity, as opposed to efficiency or ease, is her preferred way to refer to the practice of writing efficient code. When Anshika modeled efficient solutions, she almost exclusively used "I/you can just" or "all I/you have to do is" to introduce these practices. Her use of the first person as opposed to the second person appears to be approximately equal throughout both quarters, regardless of whether she is live coding or giving instructions. The one indicates modeling, the other reduces social distance. Her use of "just" and "all" are mitigations that reduce the potential face threat of advice. The modal "can" suggests permission, while "all you have to do" suggests advice. Both are low-threat ways to deliver commands to novices to choose the most efficient means possible for writing their programs.

When John instructs students to maximize efficiency, however, this directive is typically accompanied by a joke about laziness. In this example, John writes "a += 2" as a way to add two to the value stored in the variable "a," which can also be written "a = a + 2."

Greg: Isn't that "A equals A plus 2"?
John: Yeah, correct. That's really great. So this actually means, "A equals 2- A plus 2". It's like, save your time to type another A. This is, this is how lazy programmers are.

Audio-recorded in lecture: 9/30/16

Greg, a student, interjects with a correction: until now, students have been told that adding to a variable is done one way, and John has done it a different way. He mitigates the face threat

involved in correcting an instructor by inflecting the correction as a negative question. John validates Greg's statement twice using repetition and boosting ("Yeah, correct" and "That's really great"), so that by refusing his correction he does not risk confusing or discouraging Greg. Greg's formulation would be a correct one, but so is the one John originally suggested. John encourages students to use his version using justification (it will "save your time" - a concern of programmers). He attributes the value of saving time to the feature "lazy," which he applies to programmers. His reflexive positioning, connecting the negative quality "lazy" with programmers, is a ritual insult used for self-deprecation, which breaks the tension that could accompany giving advice.

John also pairs this desirable quality of laziness with "ease," for example in a section where John introduced the idea of recursion:

> John: there are some problems you can't solve- like, easily solve with the loop ((write)) like, for loop-
> Barry: -mhm-
> John: -or while loop. Then it would be a good idea to use recursion, and for now you don't actually know this, right? ((write)) This is a good idea-
> Class: ((laugh))
> John: -to use recursion. And sometimes, um, uh, I- you can just think about it like, uh, there are some times that it's a really good idea to use recursion. Because like if you use this instead of "for" or "while," mm, because there might be more lines ((write)) in the for or while, but it would be a l- much shorter to use recursion instead of just while and for. Just like for the- it's like, uh, programmers are really lazy, so like you can type as little as possible for the- any program, that would be a good solution. And this is, like, four lines, which is still a lot. So you can actually make it shorter into three lines. Does anyone have any ideas how do you do that? It's k- uh, related to what I just said about return.
>
> Audio-recorded in section 10/7/16

To begin, John frames the process of deciding what method to use as a matter of ease. The problem technically could be solved using a loop, but it would be harder than using recursion. John's definition of "easy" typically co-occurs with solutions that require fewer keystrokes or

lines of code. Here, doing less is finding an "easier" way and being "lazy," all of which is "a good idea." John is a non-native speaker of English, which could contribute to his unconventional politeness techniques. He chooses "you can" as a construction for his advice-giving. While this is not a particularly uncommon construction for giving advice, it happened to be a unique feature among the instructors. This takes a modal traditionally associated with permission, reducing the weightiness of the advice. He employs justification heavily, explaining why recursion is better than a loop in this case. Finally, his use of the self-deprecating joke further reduces the impact of the threat of advice. In general, John tended to prioritize minimizing the number of lines in a program, sometimes challenging himself to reduce lines during live coding.

John is also heavily oriented to Silicon Valley (he had had internships at tech companies and would be starting at a major tech company after his program concluded), whereas the others are interested in pursuing a career in education. His characterization of efficiency as equivalent to laziness or taking the easy path could be related to this orientation. He was by far the most likely to iconize the use of few characters or lines as a characteristic of the programmer style. Rather than reminding students to be conscientious or assuring them that they are allowed to do less in service of efficiency, he provides a humorous characterization of the culture the students can anticipate encountering.

The question of whether to change the value of a variable using the form "a = a + 2" or "a += 2" brings up multiple definitions of the concept of efficiency. In an email conversation with Nina Amenta on 14 November 2018, she described the evolution of the idea of "efficiency" as a cultural norm over the last forty years. While she considers efficiency to be a way of signaling "that someone is part of the 'in' group," her primary concept of the practice is that its purpose is

to produce better programs. The definition of "better" here is: (1) easier to write without bugs, (2) faster to write, (3) easier to understand for the reader, and (4) easier to modify later without introducing bugs. Sometimes these goals are in conflict, such as in the question described above. In the ECS 10 class, efficiency as a value should be a fundamental part of the teaching of loops, nested loops, and functions. The conflict is well illustrated by the conflicting instructions given by John and Kurt. Kurt prioritizes ease of reading by asking students to type out "a = a + 2" even though that takes more characters to do. For John, item (2) is more important: the faster to write, the more efficient the program.

Insular Nerds

One of the primary themes that emerged from the lectures served to position professional programmers as an in-group that guarded its membership through the use of gatekeeping practices and mockery of members deemed non-legitimate. For example, when a new piece of terminology emerged that Kurt considered particularly confusing, he would often introduce it by describing the term's inventors as having purposely created a situation in which novices would struggle to grasp it. For example, when describing the difference between a parameter and an argument, Kurt said:

> Kurt: Now for some really weird terminology, it was designed just to confuse you. We talked about these things as parameters. It's kind of sloppy and informal.
>
> Audio-recorded in lecture 1/25/17

Kurt correctly identified an area of confusion for his students. In both quarters, students routinely struggled to recall which term to use in which context. When defining a function, a programmer must invent names for parameters, which are written in parentheses after the function name. These parameter names will be used in the body of the function to generate some

computation. When calling, or using, a function, a student must write arguments in parentheses after the function name. Those arguments correspond to the parameters in the function's definition. By describing it as "sloppy and informal," Kurt is referring to the fact that professionals often refer to both as "arguments." Kurt's mechanism for delivering this new piece of terminology relies on his affective stance that the inventors of the terms "parameters" and "arguments" were malicious in their intent, while at the same time positioning himself as disapproving of this practice. By connecting a linguistic feature, inconsistent use of terminology, with a community trait, malicious gatekeeping, Kurt engages in iconization. Erasure, the process of ignoring certain features of a community in the pursuit of constructing a discourse that could not sustain those features, is accomplished here. Kurt describes the definitions of arguments and parameters as sloppy and informal, drawing a conclusion about programmers as sloppy and informal, which does not align with the nature of many other terms, which have extremely clear and well-defined meanings and uses. While Kurt often employs the inclusive pronoun "we" when he refers to the students, he has opted not to do so here. He has thereby created three membership categories: gatekeeping professionals, kindly professor, confused students. In so doing, Kurt reinforces a paradigm in which he is a translator between legitimate members and novices, a sort of quasi-member who could thrive among the professionals but chooses instead to lead new members into the fold.

Kurt indexed the gatekeeping professional category most often when the topic of lecture was coding style. The previous week, Kurt had introduced another evaluation of the behaviors students could expect to encounter if they entered a community of professionals. While a variable name that begins with a capital letter will not cause an error, it is considered poor style

to create one. Kurt explained this, then provided justification, evoking the threat of mockery -- more iconization -- by full-fledged members:

> Kurt: That's just, uh, something that Python programmers have agreed to do. Um, and you can get away with doing other stuff, but if you f- sort of f- fall in the- if you fly in the face of, uh, standards like that, of agreed-upon standards, if you actually start to work with experienced Python programmers, they will- they will mock you for not using, uh, lowercase letters.
>
> Audio-recorded in lecture 1/18/17

Kurt continues to refer to a tertiary schema using pronouns (I)-you-they, reinforcing a separation between himself and students, between himself and Python programmers, and between students and Python programmers. His use of the adverb "actually" further supports the presupposition that these students have not yet met an "experienced Python programmer." He positions programmers as unreasonable and unkind, a group of people who have created an arbitrary rule and will enforce it with public negative assessments.

Despite these discourses invoking stringent attitudes and mockery, on the first day of class, Kurt assured students that they did not need to be rock star geniuses like the people in history books. Kurt's description of who can be a programmer supports corporate discourses of programmers as egalitarian, while resisting those that construct them as individualistic:

> Kurt: Anyway, all this stuff and more have, have computers in them, and they need programs to be written and, and at some point, and people ask this question legitimately, you know, "Do we really need all these programs? Where do they come from?" and they come from folks just like you. It's just people that write them, nothing special, okay? Not everybody has to be Bill Gates or Steve Wozniak, or, heaven forbid, Mark Zuckerberg.
> Class: ((laugh))
> Kurt: People like you and me write these programs, so do we need more programs? We always need more programs because there's always new ideas about how this stuff, what we can do, how we can do this.
>
> Audio-recorded in lecture, 9/21/16

Kurt positions himself as similar to the students, legitimizing them as potential members of the programmer community. He refers to programmers as "people like you and me" in order to strengthen that similarity. He even engages in differentiation from one of the most famous figures in the tech industry, Mark Zuckerberg, as a move to support the discourse of egalitarianism by delegitimizing hero worship. Kurt, having suggested pair programming as the ideal intervention in the first place, is interested in addressing the mismatch between CS education environments that construct programmers as lone wolves.

Nerdy Nerds

Kurt often indexed the social category "nerd" in reference to himself, the students, and the imagined professional programmers they would eventually work with. This practice contrasts with the strong division he constructed when it came to conveying new terms or describing style conventions. Kurt sometimes constructed nerdiness by positioning himself as a nerd using science-fiction movie references, ironically referring to coolness or sexiness to index a lack of those qualities, or positioning the students as nerds using jokes. Kurt's references to nerdiness were always greeted with laughter from the class, a clear indication that his utterance was recognized as a joke. This joke constitutes ritual insult, a technique for lowering tension. Some of these instances of teasing were self-directed, as in the following example:

> Kurt: A lot of these things are so familiar to people who work in this field, they have names [. . .] Um, that's a toad. (.) You know why that's a toad? ((croak)) Think of a, think of a toad mouth opening and closing on its- I know, I know. It's just like- you- you- it takes a little nerdiness to be in this field.
> Class: ((laugh))
>
> Audio-recorded in lecture: 11/18/16

By joking, Kurt is able to take a stance on what he predicts will be seen as an absurdly silly linguistic feature of programmers. When it is time to introduce the term "toad," Kurt revoices his students' imagined objections, saying, "I know, I know." His stance is that the students are correct to evaluate the term as a silly one, but that it is indicative of the culture they are attempting to join. Rather than the tertiary schema Kurt constructs when describing the maliciousness of programmers, Kurt here creates a binary system in which he and the programmers are nerds and the imagined objecting students are not. Kurt makes no commitment that any of the students he is addressing are nerds here, but his ambiguity leaves room that they could be. Kurt nearly employs the second person pronoun "you" as the possessor of nerdiness, but instead selects the impersonal pronoun "it" to avoid committing to the proposition that he is addressing nerds. Those students who would have objected to the word "toad," then, are discouraged from voicing those objections for fear of being found out as illegitimate prospective programmers. Those who found it charming, however, are positioned as proper nerds, and therefore are reassured that they will be a natural fit among real programmers. Events such as this tended to revolve around puns or references to science fiction movies.

Some of Kurt's joking about nerdiness was directed at the students themselves, provided they meet some criterion of nerdiness. In the following example, Kurt introduces a question that will not be answered within the scope of ECS 10:

> Kurt: That really is like a- a problem for ECS 60 and beyond. Alright? So, but, like, you know, if you want to play with that in your spare time, you have nothing else to do, you know, you got a break coming up, you want to challenge yourself. You know, you have no social life, go ahead and work on it.
> Class: ((laugh))
>
> Audio-recorded in lecture: 3/15/17

Kurt refers to a feature of nerds, which is that they do not have a social life. He applies this feature to his students under the contingency that they want to "play with that in your spare time." By joking about the idea of working on a programming question in one's spare time, Kurt takes the stance that it qualifies them as nerds, and therefore acceptable members of the programmer culture. At the end of a quarter in which Kurt has been positioning himself and professionals as nerds, this move extends validation to those students who code during breaks.

Kurt also frequently made references to science fiction films of the mid to late twentieth century, such as *War Games* and *2001: A Space Odyssey*. The students tended not to laugh at or demonstrate recognition of these references, prompting Kurt to express dismay. The connection between nerd culture such as these movies and the process of programming constitutes iconization. Programmers should make jokes referencing the movies as they work, and their colleagues should ratify those jokes.

When students are constructing a sense of the social, cultural, and technical realities of their potential futures as programmers, they look to those in expert positions. The instructors represent experienced members of the community they are hoping to join, and as such, they are primary sources for discourses that describe those realities. The instructors in this course reproduce discourses that might leave students pessimistic about their fit for this field. Students may be left feeling at risk of loneliness due to unfriendly, gatekeeping members or difficulty finding common interests. In the next section, I describe students' beliefs that they are not well-suited to the life of a programmer, and their sense that negative feelings are a natural consequence of their role as ECS students.

Student Positioning

As novices, the students did not tend explicitly to position themselves as programmers, or even as nerds. They did sometimes reproduce the stance that efficiency is to be desired, however. In addition, there were several instances of students claiming that they just "didn't get" programming, or that they were not "meant" for that world, positioning themselves as a poor fit. The same student never produced both discourses, however. The most frequently referenced student identity, however, was the exhausted, stressed out student. Both students who positioned themselves as efficient and those who positioned themselves as a poor fit also positioned themselves as overworked, unhappy, or stressed out. While some of the discourses that emerged concerning experiences, there was some joy, felt when the programmer verifies that they have written good code.

Efficient

Students tended not to use the word "efficient," but they took on their instructors' use of "ease" as a metric for deciding whether a particular plan of action was desirable. For example, in lab, Rema asked Chris how she should structure a function whose job was to read through a matrix and make choices about what to assign to each position within the matrix based on what the neighboring positions contained. Chris answered by reminding her of the primary purpose of the function:

> Chris: I think, the main part of it is being able to look over (.) all of the parts of the board, right? Being able to identify when you have a 1, or when you have a 2, and then being able to look at all of the squares that are surrounding that 1 or 2, right, so being able to like-
> Rema: And for that we would probably be using the for loops to make it easier.
> Chris: Yeah, exactly.
> Rema: Instead of all of these, like, conditionals. ((laughs))

Chris: Yeah, so you're just looping through it, you're just looking at, like what possible moves you can make, and then you're creating a board based on that move.

Audio-Recorded in Lab, 3/2/17

In the place of a question about whether for loops were appropriate, Rema made a declarative statement, modified by the adverb "probably" and the conditional mood, as a request for Chris to verify whether her belief was correct. This was a typical construction for students who had relative confidence in the accuracy of their understanding. The further adjunct "to make it easier" is a bid to appropriate the stance that making things easier is a quality of good choices in programming. Chris ratifies all of this with "Yeah, exactly," a full-throated endorsement of Rema's move. Students who prioritized ease or speed were typically rewarded with agreement in this fashion in whichever context they did so.

A Poor Fit for Programming

In both lab and pair programming exercises, some students evaluated themselves as unfit for programming. Students positioned themselves as just the wrong type of person for this pursuit on several occasions while speaking to other students or TAs, but not to Kurt.

Rhea, an Asian-American woman enrolled in ECS 10 for winter quarter, felt very strongly that she was not a programmer. Rhea worked with a student who did not wish to participate in the study, so her statements are provided without those of her interlocutor. When that person asked her why she was enrolled in ECS 10, she said:

Rhea: Um, I wanted to see what programming is like. So, I was like, "ah I'll just give E.C.S. a try and see what it's, ya know, see what it's like." And, it's definitely challenging, definitely. What about you?

Audio-recorded in Pair Programming Activity, 2/8/17

Like many students, Rhea saw the class as an opportunity to test whether programming would be a viable major or minor. Also like many students, she found it difficult. Rhea positions herself as not very committed to computer science, a curious passerby. What she is committed to is the idea that it is extremely difficult for her, using boosting adverbs and repetition in "it's definitely challenging, definitely." Later, she elaborates:

> Rhea: I was actually thinking, like, maybe I would wanna try and minor in comp sci? So, that's why I was like, "I'll give this a shot," but, I mean it's cool, but it's definitely, for me personally, it's a lot to- [other student interjects] 'cause I'm like, it's just, like, so foreign to me, so foreign.
>
> Audio-recorded in Pair Programming Activity, 2/8/17

Rhea reaffirms her non-committed stance on computer science, describing her desire to minor in the subject as a "try" and wanting to "give this a shot." Rhea's identity is very clearly not aligned with that of a programmer in this session. At the same time, she does not take a fully negative stance toward programming, evaluating it positively. To her, programming is cool, but it's also "a lot" for her. Her use of boosting "so" and repetition of "so foreign" further distance her from "programmer." After working for some time, Rhea stops suddenly and proclaims:

> Rhea: ((sighs)) E.C.S. life is just not for me!
>
> Audio-recorded in Pair Programming Activity, 2/8/17

Her sigh at the end of a difficult exercise indicates that the difficulty is yet another piece of evidence to support her growing sense that she does not belong in this field. What is interesting to note is that Rhea shared with me that she earned a B+ in this class, uncurved. Despite her excellent performance in the course, Rhea is reproducing discourses that there are people who are naturally programmers and those who are not. Her use of "just" here speaks to the sense of essentialist personality traits, a belief that some people are naturally a good fit and some are not.

Several students said something like this at some point in the quarter, all of them women. Erasure, the process of ignoring or denying elements that might make a discourse untenable, is present here because these students were asking for help in well-attended lab hours, where many other students appeared to struggle as well. Believing that they were somehow uniquely struggling fit the discursive construction of themselves as specifically unfit for the pursuit of programming.

The idea that a person could be an irredeemably poor fit for this field has to do with ideologies concerning natural strengths. People are often divided into STEM people or humanities people: which category they fit into is considered essential to their personality. In the first lecture, Kurt attempted to counteract this discourse of intrinsic aptitude by telling the class that every famous innovator was once a beginner. Even armed with this knowledge, however, the idea is extremely pervasive that everyone is either naturally a good fit or a poor fit for programming. This idea was frequently reproduced in student pairs or conversations between a student and a TA.

Unhappy/Tired

A common strategy for students seeking to connect with one another was to commiserate about stress or exhaustion, an understood feature of studenthood. Students expressed the idea that they are intrinsically stressed or tired. This practice often took the form of jokes, which allowed students to position themselves as stressed-out students. For example, in this conversation between two students and myself after a discussion section, Lionel jokingly refers to crying:

> Lionel: Yeah, it's parents' weekend.
> Caitlin: Ahh.

Santiago: It is?
Lionel: Yeah
Caitlin: What do you do? Like, you just drag them around campus?-
Lionel: I honestly don't know.
Caitlin: and like, "this is Kleiber Hall, this is the social science-"
Lionel: "This is where I cry."

Audio-recorded in section 11/4/16

I begin by asking, "What do you do?", a stance that the prospect of bringing parents to campus is absurd, that there would be nothing of interest to do there. I further add a joking suggestion, using the hyperbolic verb "drag" to describe making them walk to essentially identical areas of the school for a day. Elaborating, I dialogically voice his future self, repeatedly naming buildings to illustrate the dullness of walking them through campus. Lionel then takes up my dialogic voicing, providing a further example of what he might say. Lionel's use of joking in "this is where I cry" positions him as a stressed-out computer science student, who cries because of his work but continues to do it. Whether this positioning is a bid for a programmer identity or a student identity in general is unclear. What is clear is that this is a practice that was not modeled by the instructors.

This positioning practice was prevalent not only in after-section contexts but also in lab and pair-programming activities. For example, Ernie and Brigitte ended their first pair programming activity with a brief conversation about how worn out they were. Like Lionel above, Ernie positions himself as a stressed out student. When the first pair programming activity is about to conclude, he and Brigitte consider working on the problems together at a later time. Brigitte has a class to go to that afternoon, but she wonders if Ernie is available afterward:

Brigitte: Um, if anything, ah- what time do you finish class?
Ernie: I'm done ((laughs)).
Brigitte: You're done?
Ernie: Um, I'm- I'm done at 12 every day.
Brigitte: ((sighs)).

Ernie: Yeah I'm just tired ((laughs))
Brigitte: Um, I don't know.

Audio-recorded in Pair Programming Activity, 2/27/17

Ernie employs a play on words to harness the double meaning of "done" - he is done with classes for the day, but he is also just done, as in uninterested or unable to continue this task. He is taking on the student identity of total depletion, a cultural reference that Brigitte eventually recognizes. Brigitte may feel she has more license than Ernie to position herself as overly stressed, as she does not have as forgiving a schedule as Ernie does. Her sigh could be seen as an expression of that stance, a non-endorsement of Ernie's positioning move. It could also be seen as a commiserating act in which Brigitte accepts Ernie's positioning and aligns herself with it as well. The statement "Yeah, I'm just tired" is an interesting move after the declaration that he finishes classes at noon every day. It suggests that his tiredness is not the direct result of his course load; rather, it is a feature of his identity. He is not tired because of anything, he is just tired. Being tired or stressed out may be seen as a feature of studenthood, which Lionel, Ernie, and Brigitte just are, by dint of being students.

According to the students of ECS 10, being a programming student is a highly unpleasant thing to be. For those who feel they are a natural fit for programming, efficiency is an explicitly stated goal. For those who do not, a sense of poor fit pervades even successful programming events. For any student, the life of a programming student is characterized by sadness and exhaustion. It is reasonable here to draw a connection between Kurt's discourses of insular and unlikeable nerds and students' concerns about fitting in. Whether that link is causal is not obvious, but classrooms could benefit from incorporating more explicit attempts to counteract discourses that might reinforce student fears.

Appreciative of Good Code

Students and instructors alike expressed joy as part of their programming experience. While coding itself was often associated with stress or frustration, the successful execution of that code was framed as worth the negative feelings. As part of their propensity to encourage efficient code, the instructors modeled a sense of admiration for "beautiful" code. Similarly, they expressed exuberance and relief when their code was functional. For example, when Kurt finished a live coding demonstration of writing a while loop, he ran it and then celebrated its success with an imitation of a trumpet:

> Kurt: All right. (.) Over on this, we would like to save it, okay. ((run module)) While demo. Oh yeah yeah. (.) Look at that! ((mouth trumpet fanfare)) What? Nothin'?
> Some: ((clap))
>
> Audio-recorded in lecture, 10/21/16

When he runs the module, Kurt demonstrates the functioning code on the projected screen. By engaging in an outward expression of jubilation, Kurt models the unbridled sense of excitement he considered a natural part of testing code. When programmers test code and it works, they feel so excited that they must celebrate. Kurt jokingly asks, "What? Nothin'?" taking the stance that he is surprised the class has not joined him in his celebration. This stance is taken jokingly for two possible reasons: the first is to demonstrate that programmers feel excitement when code is functional; the second, to show the students that joy is a significant part of the life of a programmer. The students ratify his stance with applause.

This joy at functional or well-written programs was reproduced several times during the pair programming assignment. For example, when Brigitte and Ernie had been working on a problem for a while, having negotiated several tricky items, Ernie encouraged Brigitte to try a technique she had not been entirely comfortable with. When she did it, and the program executed correctly, the two of them became extremely animated:

Ernie: ((laughs)) This- er, this- we have too many!
Brigitte: But it works! ((laughs))
Ernie: Yeah, it does work.
Brigitte: It's just the most beautiful thing I've seen in my life! ((laughs)) Ah! That is so beautiful.

Audio-recorded in Pair Programming Activity, 2/27/17

Brigitte had been expressing both confusion and frustration, but when the program executed successfully she was overcome with positive feelings. Brigitte's variation in intonation and stress patterns increase noticeably when it becomes clear that the program works. She uses "just" here to suggest that the beauty of the functional code is inarguable. She uses repetition of the word "beautiful" to boost her meaning. This sense of total happiness comes at the end of a challenging and often frustrating process, which frames it as a kind of reward for the negative feelings associated with the programming student identity.

Gender, Race, and Student Success

As part of their participation in this study, students answered a questionnaire in which they described themselves in terms of race and gender identities, then answered questions about their experiences in the course. The students also gave me access to their course grades as part of the university registrar release. In this section, I describe the quantitative relationships that held significance in terms of student grades and whether or not they went on to take more computer science courses after ECS 10.

In order to examine the association between students' race, gender, and final grades, I used multivariate analysis to describe these variables in relation to each other. A preliminary analysis revealed no significant relationship between students' declared race and their final grade in the course, except for one group: Tukey's HSD revealed that East Asian students

outperformed Latinx students (p = 0.0005) in both quarters. I was unable to measure whether Black students experienced the same difference in score because there were no Black students enrolled in Winter Quarter, and only three were enrolled in the Fall. Latinx and Black students could be a subject of deeper study in the future. As for this study, I have no significant results to report in terms of how race might interact with the potential benefits of pair programming.

As a predictor of grades, gender was significant in Fall but not in Winter. Tukey's HSD revealed that in Fall Quarter, men performed significantly better than women (p = 0.014). In Winter Quarter, however, they did not (p = 0.642). In either case, the difference in final scores was small, if significant. Table 5 contains the final score of students who released their registration information, arranged by the quarter in which they enrolled in ECS 10 and the gender they indicated they most identified with in the questionnaire.

Table 5: Final score by self-reported gender and quarter enrolled

Quarter	Mean Score			Tukey Pr(>F)
	Men	Women	All	
Fall (no pair programming)	86.43	83.36	85.24	0.00017
Winter (pair programming)	84.87	85.19	84.91	0.0016

Where the scores for men were slightly lower in winter than they were in fall, women's scores rose. Speaking to the instructors, I asked if there were any major differences between a student who took ECS 10 in the fall and one who took it in the winter. They told me that they expected grades to be lower overall in the winter for several reasons. Winter quarter is considered challenging for all students, no matter what courses they are taking. The instructors

were not sure of the reasons for this, but Kurt suggested (and I agree, looking back on my time as an undergraduate at a major university) that burnout and gloomy weather could be factors. Regardless of cause, the small but significant rise in women's grades is even more surprising given the expected drop as a result of being a student in winter quarter. We can take this effect as a potential argument for pair programming as a teaching tool.

A small number of participants revealed whether they took an ECS course the quarter following ECS 10. This question was optional, so 94 students answered in Fall Quarter and only 44 in Winter Quarter. For those students who chose to do so, I also measured the relative significance of various factors in their decision. Gender ($p < 0.001$) and ECS 10 grade ($p = 0.001$) were both significant factors in students' decisions to continue in the program, while race was not. Table 6 illustrates the proportion of women who chose to take another ECS course in the following quarter, for ECS 10 students in the Fall and in the Winter.

Table 6: Choice to take more CS courses by gender and quarter

	Fall (Control)		Winter (Treatment)	
	Did not continue	Continued	Did not continue	Continued
Women	15	27	20	7
Men	3	49	4	13
% Women	83.3%	35.5%	83.3%	35%
Total	18	76	24	20

The use of pair programming in the Winter quarter was primarily aimed at determining whether it might help students decide to continue their studies. Gender was more significantly a

factor in Fall ($p < 0.001$) than Winter ($p = 0.002$), which means that Winter students' identification as women was slightly less likely to negatively impact their decision to stay. Women comprised approximately the same proportion of respondents who took another ECS course in each quarter. All the measurements undertaken resulted in very subtle differences between the treatment and control groups.

As in the case of student grades, plans to take another course the following quarter were expected to decrease for Winter Quarter students. In several conversations with Kurt, he informed me that students who were strongly considering becoming ECS majors were much more likely to take the course in fall so that they could move forward in the course sequence right away. Students who took ECS 10 in the winter were more likely to be fulfilling course requirements for a major outside of the ECS department. This is supported by the fact that fewer students took ECS 10 in the winter than in the fall. This attitude constitutes a systematic difference between the groups and may provide some insight in the lack of change between quarters. With no intervention, it should be expected that the students' grades and plans to take another ECS course should be lower in the winter; this year, they were not.

Conclusion

As they communicated their beliefs to students about the kind of programmers they could expect to encounter in the workforce, instructors tended to deliver a relatively bleak view of the future. Discourses describing programmers as unkind and unlikeable may have left students concerned that they would be socially unfulfilled or even made miserable by their fellow community members, if they joined a company as a programmer. One core value of programmers, efficiency, was communicated via several diverse methods, but it appears to have

been taken up by several of the students. It was also not typically accompanied by these discourses about what a programmer's personality was like, which may have contributed to students' relative ease of uptake. Some students, particularly women, who encountered difficulties, tended to take those difficulties as a sign of their natural poor fit for the field, falling into erasure and essentialism to strengthen their beliefs. Increasing the amount of pair work students had to do may have decreased the effect of that tendency, as gender had less to do with grades and retention in the treatment quarter than in the control, and grades and retention rates did not drop off as expected in winter quarter.

CHAPTER 7: CONCLUSION

In this chapter, I summarize the research questions driving the current study, contextualizing the relevant data with the literature pertaining to those findings. I describe the ways in which the research produced for this dissertation contribute to the literature that came before it, and discuss the implications for the Language Socialization research community as well as educators interested in managing the socialization of new members to CS, STEM, or other technical fields. I discuss the limitations of this study, and I propose activities which might ameliorate those limitations and future study that can build upon the findings contained in this dissertation.

Research Questions

In the first chapter of this dissertation, I outlined several research questions, to which the answer was addressable by some combination of participant observation, recorded pair programming activities, and experimentation. This study was broadly concerned with the performance of students, particularly those who identify with underrepresented groups, as a function of interlocutor type. In this section, I list those questions once more and summarize the relevant data and literature in an effort to answer them.

In what ways do instructors socialize students to use programming-related terminology? In what ways do students take up those terms?

The instructors' use of Python and programming terminology appeared to take a form recognizable as modeling. Kurt repeated terms in several contexts and demonstrated different syntactic and morphological features of those terms. The TAs did this as well, but they tended to

do so with less frequency, and they modeled fewer of the appropriate inflectional and derivational morphemes available to full members in the community. Repetition also achieves the goal of demonstrating the parameters of the word's coercion into spoken English (Duke, 2017). Repetition is considered a useful strategy for introducing new terms (Moore, 2011), but it is also one of the positive politeness strategies listed as a way of seeking agreement with a hearer and thereby restoring some positive face for them (Brown & Levinson, 1978). Building on research that suggests that face wants are discernable from conversational analysis and that doing so can create a picture of interpersonal relations (Arundale, 2010), Brown and Levinson's (1978) strategies for face threat redressive strategies.

Kurt and the TAs all employed face threat redressive strategies when modeling terms. Several of these strategies fall under Brown and Levinson's (1978) positive politeness goal, "claim common ground" (p. 102). One strategy they employed toward this goal was Brown and Levinson's Strategy 7, presupposing H's agreement. One way to achieve this is to use personal-centre switch, a type of point-of-view operation: through the use of inclusive "we," positive face is reinforced. Kurt and all the TAs employed Strategy 6, "avoid disagreement," when correcting their students. They tended to do this through token agreement, agreeing to all or part of an incorrect statement as a way to segue to a statement describing the desired answer. Another way for all the instructors to manage potential face threat when correcting a student was positive politeness Strategy 13, giving reasons, also referred to as justification. Strategy 8, joking, falls under the same general positive politeness goal of claiming common ground. Joking was used mainly by Kurt and John, and sometimes by Chris. It was not a common strategy for Anshika, however. Kurt was the only instructor to use Strategy 3, "intensify interest to H" on a consistent basis through the use of storytelling. Instructors also used on-record negative politeness

strategies when introducing new concepts or correcting students, such as hedges on the illocutionary force of those acts. Tag questions are one such strategy, as they flout the maxim of manner, creating a question where a declarative was expected (Brown & Levinson, 1978). Hedging on a Gricean maxim allows instructors to be maximally direct with some nod to redressing potential face threat.

One of the ways in which students were able to demonstrate uptake was by asking for clarification. This is because nearly all the conversational opportunities for students had the stated goal of providing assistance to them. These opportunities included approaching instructors after class and attending lab hours. Asking questions of instructors involved considerable use of face threat redressive strategies, indicating a significant concern about the perceived weightiness of such speech acts and fear of losing positive face through misuse of class terms. To the first point, students tended to prefer off-record requests, asking for help through the use of problem statements. Problem statements are a way of indicating a need without overtly asking for the hearer to meet that need; it is part of Strategy 1in Brown and Levinson's (1978) list of negative politeness strategies toward the goal of inviting conversational implicatures by appearing to violate the conversational implicature of relevance (Grice, 1975). Students also used honorifics, which is Strategy 5 toward minimizing an imposition (Brown & Levinson, 1978). To the second point, students tended to avoid using terms when speaking to instructors. They did so using deixis, pauses, and gesture. For all students who requested assistance, there were several instances in which a term would be expected but one of these three events took place instead. Avoiding terms was supported by the instructors, who chose not to require students to articulate the issue using terms after the speech act was accomplished. In lab, TAs sometimes even typed on behalf of the students. These practices reduced the number of opportunities students had to

construct and demonstrate their evolving expertise in terminology use. The instructors' avoidance of commanding students to use the relevant terms when asking for help may have been the result of sensitivity to the students' face wants. In math tutoring contexts, this desire to avoid threatening students' positive face by placing them at risk of saying something incorrectly can be a hindrance to learning (Person et al., 1995).

What is the effect of increasing student collaboration on classroom talk, particularly the ways in which students use terms and achieve important speech acts relevant to learning?

Creating a designated activity in which students were required to speak to one another significantly increased the amount of interaction the students participated in. This change even expressed itself in contexts not devoted to pair programming. In Fall Quarter, the control group, students almost never sat together in lab, preferring instead to sit with spaces between them. At the end of the quarter, when labs were much more populated, students formed small groups. In Spring Quarter, students formed groups from the beginning of the quarter. This difference may be at least partially attributable to the routine addition of pair programming activities in that quarter. Students in that quarter were more likely to use class terms while speaking to one another, actively socializing each other to converge on the proper usage that had been modeled by their instructors. Student use of class terms was so rare in the first quarter that they did not have the chance to demonstrate this gradual increase in expert-like speech. The beginnings of this process were evident in Kurt's refusal to repeat or support non-expert use of terms. Reasons for this difference between the two quarters may have been a result of student fears about misusing terms in the presence of experts, reticence to demand the time and attention of those experts, or a combination of the two. Evidence that students feared the positive loss of face

includes the discourse markers that indicate significant concentration or trepidation while speaking. Students asking for help used many more repairs, pauses, and filler markers such as "like" when they were speaking with instructors. They demonstrated several types of disfluency to indicate tentativeness. Students avoided using class terms when asking for help, and they did not resist the conversational turn-taking structure in which instructors spoke in long, complex turns and students mainly backchanneled. When asking for help, students used minimizing language, which is Strategy 5 in Brown and Levinson's (1978) list of negative politeness strategies toward the goal of avoiding the sense that the speaker is coercing the hearer to perform the request (Brown & Levinson, 1978). Students asked to produce answers to questions posed by instructors used high-rising terminal and question words to indicate uncertainty. Because American English prosody uses a higher pitch at the end of phrases to indicate questioning or an anticipated continuation of ideas, this behavior during what should be a declarative speech act suggests that students are taking the epistemic stance of uncertainty using a prosodic hedge (Brown & Levinson, 1978). Epistemic stance is a part of participation in a community of practice (Chafe & Nichols, 1986) and communicates an aspect of student identities at play.

As a comparison to peer dyads, peer talk involved faster, more even conversational turn-taking than did talk between students and instructors. Students speaking to each other were more likely to laugh than students speaking with instructors. Students were much more direct with each other, suggesting a significantly lower weight to any potential face threats (Brown & Levinson, 1978). Peer talk also involved more student use of terminology than did talk between students and instructors.

What discourses and identities do instructors construct as they socialize their class as members of a culture of programmers?

The instructors performed nerdiness in several different ways, by positioning themselves as highly interested in computers and other technical activities, by making references to science fiction popular culture, and by referring to a culture of insularity and intense judgment concerning unratified members. The programmer identity constructed appeared to take elements from nerd culture described by other research (Kendall, 2000, e.g.). Most of the popular culture references were made by Kurt and had to do with films from the 1980s with which the students were not familiar. Language socialization is the socialization to use technical language, but full membership in a community of practice also requires the ability to participate in other domains (Roberts & Sarangi, 1999) including pop culture references (Mertz, 2007; Duff, 2010). Discourses describing programmers as judgmental and willing to mock those who do not demonstrate expert style may have led students to feel concerned about their likelihood of belonging in the community. A sense of belonging is important for helping students persist in STEM (Goodenow, 1993; Veilleu et al., 2012). Other research in similar fields suggests that a sense of belonging can be positively impacted by ensuring that instructors convey a wider range of identities for programmers in the world (Schulte & Knobelsdorf, 2007; Zander et al., 2009; Wong, 2016; Hansen et al., 2017). Nerdiness, especially programmer nerdiness, is associated with an all-encompassing desire to be on a computer at all times. While instructors did not model the discourses that there is an innate fitness for programming that each person either does or does not have, those discourses are associated with nerd identities, which instructors did model. Instructors also modeled efficiency as a strong value for members of programming communities. Discussions of efficiency included positioning it as a way of minimizing student effort, a way of

ensuring good design, and a way of creating beautiful code. This was done through the use of joking, permission-giving, and modeling actual coding practices. Positioning themselves as efficient was a way of claiming expertise while also allowing students to be engaged in legitimate peripheral participation (Lave & Wenger, 1991). The instructors, particularly Kurt, also modeled expressions of joy when code worked. These expressions included laughter, joking, and interjections. Students were socialized to take this practice on through affective stances that suggested the expectation that they will feel emotionally invested in the quality of their work. Discourse studies of American and British programming communities have found that programmer culture is associated with a sense of joy at working on computers (Hapnes & Rasmussen, 1998; Kendall, 2000; Margolis & Fisher, 2002; Leder & Vale, 2004). It is part of the nerd identity that is so tightly bound up in the participants' concept of what it is to program.

In what ways do those students support, reconstruct, or resist those discourses and identities?

The instructors performed nerd identities in various ways, and sometimes made reference to the discourse that programmers are nerds. While the instructors did not actively support the discourse that programmers are people who innately understand how to program, this discourse is documented in the wider world (Kendall, 2000, e.g.). Students in this study appeared to believe it was possible for the entire field to simply not be a fit for them. This belief suggests that there is some propensity to programming that is innately present in some people and absent in others, and no amount of studying or practice will make a non-programmer into a programmer. This discourse is also connected to those that associate nerd identities with computer science. This means that when instructors constructed the nerdy identities as the best fit for programming, they may have inadvertently supported the damaging innateness ideology in so doing. "Nerd" is

a cultural concept associated with masculinity (Kendall, 2000; Clegg, 2001; Oldenziel, 2001; Margolis & Fisher, 2002). It is also considered innate, a quality apparent in early childhood and is something people engage in all the time, with no sense of moderation (Hapnes & Rasmussen, 1998; Margolis & Fisher, 2002). Women particularly appeared to support discourses of innate programming prowess and naturalness of fit.

There were several ideologies, stances, and identities that students took up with some modifications. Students appeared to take on ideas of efficiency as important practices for members of programming communities of practice. They did not reproduce this idea exactly, instead prioritizing "ease" while they discussed code with instructors and peers. In language socialization, novices may not precisely copy the ideologies or practices of experts, but they are still engaging in socialization when they do modify linguistic practices (Ochs, 2000; Kulick & Schieffelin, 2004; Morita, 2009; Goodwin & Kyratzis, 2012; Rogoff et al., 2014; Ochs & Schieffelin, 2014). Students seeking to connect with one another positioned themselves as emotionally or physically depleted. This practice often took the form of jokes, which allowed students to position themselves as stressed-out students. It is possible that this stance of abject misery is a subtle way in which students took up the nerdy identities: rather than choosing to spend every waking hour in front of the computer, they do it because they must. This would constitute another modification of socialized practices on the part of novices. Misery appears to be a prevalent affective stance for students to take, and the way for them to get relief from that feeling is by taking up the linguistic practice of expressing joy, as modeled by their instructors. Students also took up expressions of joy at code they judged as beautiful or at least functional. The affective stance student took echoed the belief they observed in their instructors that beauty is a possible feature of code. When students got the result they desired, they reacted with

laughter, joking, and interjections, very similarly to Kurt's expressions. As discussed in the previous section, taking joy in the programming process is considered a part of the nerd identities that exist in the larger cultural context, although joy at the functioning of code despite misery while writing it is not.

What is the statistical relationship between quarter (control group versus treatment group), academic performance, and plans to continue in the program?

Because of various methodological challenges involved in the execution of this study, statistical results are limited in what they can tell us. It is likely that women earned higher grades and had more intention to continue to take CS courses as a result of their enrollment in winter as opposed to fall quarter. It is possible that students of color did as well, although this is only verifiable for Latinx students, and in their case there were not enough Latinx students to be certain of the validity of these results. The measurement of relative performance for Black students was not possible because there were no Black students enrolled in Spring Quarter. The measurement of these phenomena to a significant degree of certainty is not possible due to the limitations discussed later in this chapter.

Contribution to the Literature

In 2017, an argument was presented in the journal *Computer Science Education* that identity formation was a necessary and missing part of the national discussion about representation in CS (Rodriguez & Lehman, 2017). The authors of this article summarized research into identities and socialization relating to school subjects related to CS, but these subjects were not similar enough to CS to produce recommendations that could create lasting and

effective change in CS programs. This study attempts to accomplish the goals stated by Rodriguez and Lehman and by Professors Amenta and Eiselt in meetings with me. By using Language Socialization Theory and its related analytical methods, this study is able to address issues of identity, inclusion, and cognition, making it one of the very first of its kind, and the first of this depth and breadth of inquiry.

Implications

In this section, I will discuss ways for departments to increase the degree of language socialization that takes place in an introductory course. At UC Davis, graduate students who will be assisting in CS courses are required to attend a course devoted to learning how to be a teaching assistant. This course primarily addresses classroom management and use of the technology in the classroom rather than pedagogical practices. A significant impact could be made by teaching those graduate students about the following recommendations.

Repetition is a method used frequently by Kurt and recommended by the literature as a way of maximizing the modeling stage of Language Socialization. TAs should prioritize repetition when introducing new terms. The TAs who participated in this study demonstrated a much higher use of deixis and gesture where the term could have been used than did the instructor. TAs could be shown the transcribed example from Kurt's lesson in Chapter 4 as a target. In their training class, they could then be asked to speak about a particular CS topic for a period of time, using a target term as many times as possible.

Instructors should refuse to ratify non-expert term use by repeating the student's utterance with the terminology used properly. Instructors should also refuse to support avoidance and wait for students to articulate their issues using the terms. This will feel uncomfortable for

those TAs because of their instinct to use positive politeness strategies such as Brown and Levinson's (1978) positive politeness strategy 6, avoid disagreement. All instructors would benefit from practice engaging in uncomfortable face-threatening acts in order to ensure they do not support students' avoidant behaviors. One strategy they can use to mitigate this threat is scaffolding. The instructors could be encouraged to begin the students' sentences for them and wait for them to provide the most academically crucial words. This practice would serve the dual purpose of reducing the trepidation exhibited by students and socializing them further into the professional discourse of programmers that has been shown to include scaffolding.

While humor is a reliable tool for reducing social distance and improving student comfort, comedic references to nerdy pop culture can create unintended negative results. These references can remind students that there is a stereotype of programmers that does not fit their self-concept. Because this culturally understood identity is thought to involve all-consuming love for programming, students who have other interests, or whose desire to be a member of a programming community is less passionate, may feel they will not find success or happiness in this field. Similarly, instructors should avoid jokes about being judged harshly for violating coding style conventions. These jokes point to discourses about exclusionary and highly judgmental expert members, which reminds students of the jealously guarded technical masculinity present in the culture. Jokes about being miserable, overly stressed, or sleep-deprived also contribute to students' sense that joining a community of programmers will result in feeling depressed and overworked if they embark on a career in CS. Humor about student culture or the silliness of certain programming terms (such as Kurt's joke about the frog's mouth) are a way to ensure that joking is present, as it is a valuable tool for instructors.

Instructors should make more of an effort to explicitly counteract the pervasive belief that programming abilities are innate and nerds are the only people who can program. Kurt spoke to the students once about this, but it appears they continued to believe it throughout the course. The short speech Kurt gave on 21 September, 2016 contained many positive elements: it involved humor, he positioned himself and the students to the same identities, and it differentiated regular, successful programmers from tech heroes such as Steve Wozniak. This type of talk should be repeated more often and by more instructors in order to subvert the belief that tech prodigies as the only type of person who can make it in this field. Those who supported this ideology most were women, who do not fit the cultural concept for a typical programmer. These students in particular need extra care as they risk exclusion most. Explicitly engaging with damaging discourses has been shown to be effective in reducing the effects of stereotype threat and other forces that keep underrepresented members from pursuing full membership (Gordon, 2004). One way of achieving this would be to spend some time in the introductory course studying research that describes discourses of programming. Some of the studies cited in this dissertation would be a reasonable place to start, including workplace discourse studies and popular culture analyses.

Many of the discourses the instructors supported were somewhat different from the harmful ones discussed in Chapters 1 and 2. While discourses of natural fit are of concern when it comes to the TAs' speech, the instructors' tendency to foreground efficiency and learnable best practices lends support to discourses that programming is a set of practices that anyone can learn. Instructors should be told that this practice, while already present, could be foregrounded, and that discourses connecting efficiency with ease of work should be encouraged. This includes rewarding the tendency TAs showed to joke about themselves as lazy. A positive addition to the

TA training course would be an explicit discussion of the values they would like students to have. One important aspect of this would be to discuss the four dimensions of "efficiency": bug-resistant design, speed of writing, ease of reading, and ease of modification. When these dimensions conflict, which should take precedence? This discussion could help avoid some contradictory advice found in the instructors' data.

Kurt spoke some about his home life and children in class - a reminder that he is not a single-minded programmer helps combat discourses that true programmers have to be obsessed with programming at all times. Professors and TAs could increase the frequency of their references to a happy and well-rounded life. A positive addition to this course could be the addition of guest speakers who can show that working in CS fields can be enjoyable, and that it is possible to work in CS and have a well-rounded life. This would give the instructors a chance to show that programmers can be happy, successful, and charming people. They could speak about what they do at work and why they enjoy it, and describe what their work and social life is like. While the presence of role models who do not seem social or happy is detrimental to the work of increasing the involvement of underrepresented people, speakers who identify with those underrepresented people, and who can successfully describe their jobs and lives in attractive ways, can have significant effects on students' plans to stay.

Limitations and Future Directions

While this study provides significant insight into the socialization process by which identity formation and terminology acquisition is achieved, there are some limitations. Due to the high number of participants, an in-depth profile of each learner is not possible. A future study could deepen our understanding of these processes by focusing on a small number of

underrepresented students and seeking to find the extent to which the language practices, identities, and stances performed by instructors have been taken up. This study attempted to uncover the linguistic practices of students in all of the school contexts, but it did not extend to what students did at home. While I did not find evidence that students were working together outside of class, it is possible that they did so without informing me. A study that prioritized interviews more than this one could draw meaningful conclusions about peer socialization outside the school.

The two quarters have been discussed and compared as a control and treatment group. However, there may be systematic differences between the two. Students who enroll in introduction to programming in the fall may perform differently and have different goals than those who enroll in the winter. The time of day and day of the week may also have impacted student performance: some sections took place at 8am, some at 5pm. A college student may function very differently at these two times. This creates some question as to the validity of the statistical measures described in Chapter 6. A more robust experimental design would have two sections with each TA at similar times, in which one section completes pair programming tasks and the other does not.

Future study could focus more on interviews to create a case study of particular students of interest. There could also be investigations of other speech acts and their socially understood pragmatic requirements within this context. Because of the dearth of research on language socialization in CS classrooms, more contexts should be explored. This includes university courses, but also courses for younger and older students alike. I look forward to a comparison between CS as it is treated in public schools and how students at the increasingly prevalent STEM private and charter schools are socialized to become programmers. There is a significant

amount of knowledge concerning how novices begin to join this particular community of practice, and it is important to learn more as programming becomes a more significant part of life for those entering the workforce.

Conclusion

Having grown up among software developers, and having studied programming myself in several educational contexts, I hoped to explain why so few women and people of color occupied the technical positions at the companies I knew about. When I described my planned research to my father, who has been a software developer for forty years, he told me that I was unlikely to find anything because people who are good at programming just get it, and people who do not have an innate talent cannot learn it. I hoped and believed he was wrong, because believing in the innateness of programming skills would suggest that there is some natural deficiency in underrepresented groups. I believe that by documenting the processes by which cultural concepts about who can be a programmer are conveyed, this research could contribute to our understanding of how to ensure that newcomers to the field are not unduly excluded. This same documentation can also allow educators to make meaningful changes to their approach as they ask students to learn new concepts and practices relating to computer science.

APPENDIX A: TRANSCRIPTION CONVENTIONS

?	Rising intonation
.	Falling intonation
(.)	Pause
-	Interruption
(())	Sound or gesture

APPENDIX B: QUESTIONNAIRES

Student Questionnaire

Please answer any and all questions to the best of your ability. If any question makes you uncomfortable or feels too invasive, or if it does not apply to you, feel free to skip it.

What is your name (Last, first)?

Who is your TA?

- Tom
- Chris
- Anshika
- John
- Gwen (Fall only)

What do you like and/or dislike about your TA's style of teaching?

What do you like and/or dislike about Professor Eiselt's style of teaching?

Which class-related activity do you find most helpful?

- Going to lab
- When the TA or professor explains concepts verbally
- When the TA or professor codes on the projector
- Reading the textbook
- Completing the homework assignments
- Pair Programming activities (Spring only)

Why did you choose the option in the previous question?

How many times did you attend section this quarter?

How many times did you go to lab this quarter?

How old are you?

For the following questions, students were asked, "To what degree do you agree with the following statement?" and given the options:

Strongly disagree, Disagree, Neutral, Agree, Strongly agree

- I feel I can succeed in the field of programming.
- I plan to major in ECS or CS.
- I felt the other students in this class were more prepared than I was.

My instructors made me feel silly or uninformed.

I felt that the other students in this class were similar to me.

I felt that I was unlikely to find friends in this class.

My instructors appeared interested in helping me learn to program.

When I started in ECS 10, I was experienced in programming.

In what level of education are you currently enrolled?

Freshman

Sophomore

Junior

Senior

5th Year

Graduate (Masters)

Graduate (PhD)

Other

What is your major? (drop down menu of all majors offered at UC Davis)

What is your minor?

With what gender do you primarily identify?

Man

Woman

Transman

Transwoman

Transgender

Genderqueer

Genderfluid

Other

When people meet you for the first time, how do they usually perceive you?

Man

Woman

Transman

Transwoman

Transgender

Genderqueer

Genderfluid

Other

With what race do you primarily identify?

White

Black

Asian

Latinx

Pacific Islander

American Indian

Other

When people meet you for the first time, how do they usually perceive you?

White

Black

Asian

Latinx

Pacific Islander

American Indian

Other

Before taking ECS 10, what type(s) of experience had you had with programming?

None

Classes in high school

Classes in college

Camps

School clubs

Clubs outside of school

Online tutorials

Self-taught

Other

What language(s) did you learn in these settings?

Which ECS courses have you taken/are you taking since completing ECS 10?

I did not complete ECS 10.

I did not take another ECS course, and I do not intend to.

I did not take another ECS course, but I intend to.

ECS 12: Introduction to Media Computing

ECS 15: Introduction to Computers

ECS 20: Discrete Mathematics for Computer Science

ECS 30: Programming and Problem Solving

ECS 40: Software Development and Object-Oriented Programming

ECS 50: Computer Organization and Machine-Dependent Programming

ECS 60: Data Structure and Programming

Other

REFERENCES

Airenti, G., Bara, B.G., & Colombetti, M. (1993). Conversation and Behavior Games in the Pragmatics of Dialogue. *Cognitive Science,* 17, 197-256.

Allen, A. & Thomas, T. (2006). Gender Differences in Students' Perceptions of Information Technology as a Career. *Journal of Information Technology Education,* 5(1). 165-179.

Alvarado, C. & Dodds, Z. (2010). *Women in CS: An evaluation of three promising practices.* In Proceedings of the 41st SIGCSE technical symposium on Computer science education, 57, 61.

Anderson, F. (1995). Classroom Discourse and Language Socialization in a Japanese Elementary-School Setting: An Ethnographic-Linguistic Study, Doctoral dissertation, University of Hawai'i.

Atwater, M.M. (1996). Social Constructivism: Infusion into the Multicultural Science Education Research Agenda. *Journal of Research in Science Teaching, 33*(8), 821-83.7.

Austin, J.L. (1962) *How To Do Things With Words: The William James Lectures delivered at Harvard University in 1955.* Oxford: The Clarendon Press.

Bakhtin, M. (1981). The Dialogic Imagination: Four Essays. Austin, TX : University of Texas Press.

Baquedano-López, P. (1997). Creating social identities through doctrina narratives. *Issues in Applied Linguistics, 8*(1), 27–45.

Baquedano-López, P. (2000). Narrating community in doctrina classes. *Narrative Inquiry, 10*(2), 429–452.

Baquedano-López, P. & Kattan, S. (2008). Language Socialization in Schools. In P.A. Duff & N.H. Hornberger (Eds.), *Encyclopedia of Language and Education, 2nd Edition, Volume 8: Language Socialization* (pp. 161-173). New York, NY: Springer.

Barker, L.J., McDowell, C., & Kalahar, K. (2009). Exploring factors that influence computer science introductory course students to persist in the major. In *ACM SIGCSE Bulletin,* 41(1), 153-157.

Bauman, R. (1986) *Story, Performance, and Event: Contextual Studies of Oral Narrative.* Cambridge, UK: Cambridge University Press.

Baxter, L.A. (1984). An investigation of compliance-gaining as politeness. *Human Communication Research,* 10(3): 426-456.

Beach, W.A. (1993). Transitional regularities for 'casual' "Okay" usages. *Journal of Pragmatics,* 19. 325-352.

Bell, D.C., Arnold, H., & Haddock, R. (2009). Linguistic Politeness and Peer Tutoring. *Learning Assistance Review.* 14(1). 37-54.

Ben-Ari, M. (2004). Situated Learning in Computer Science Education. *Computer Science Education, 14*(2), 85-100.

Benke, E. & Medgyes, P. (2005). Differences in Teaching Behaviour between Native and Non-Native Speaker Teachers: As seen by the Learners. In Llurda, E. (Ed.) *Non-Native Language Teachers: Perceptions, Challenges and Contributions to the Profession,* (pp. 195-215). Springer: Boston MA.

Berenson, S.B., et al. (2004). Voices of Women in a Software Engineering Course: Reflections on collaboration. *Journal of Educational Resources in Computing,* 4(1).

Beyer, S. (2014). Why are women underrepresented in Computer Science? Gender differences in stereotypes, self-efficacy, values, and interests and predictors of future CS course-taking and grades. *Computer Science Education,* 24(2-3). 153-192.

Beyer, S. & Haller, S. (2006). Gender differences and intra-gender differences in Computer Science students: Are female CS majors more similar to male CS majors or female non-majors? *Journal of Women and Minorities in Science and Engineering,* 12. 337-365.

Beyer, S., et al. (2003). Gender differences in computer science students. *Paper presented at the 34th SIGCSE Technical Symposium on Computer Science Education*, 35(1). 49-53.

Bhardwaj, J. (2017). In search of self-efficacy: development of a new instrument for first year Computer Science students. *Computer Science Education,* 27(2). 79-99.

Birner, B.J. (2013). Introduction to Pragmatics. Malden: Wiley-Blackwell.

Björkman, C. (2005). *Crossing boundaries, focusing foundations, trying translations: Feminist technoscience strategies in computer science* (Doctoral dissertation). Karlskrona: Blekinge Institute of Technology.

Blom, J.P., & Gumperz, J.J. (1972). Social meaning in linguistic structures: Code-switching in Norway. In J.J. Gumperz & D. Hymes (Eds.), *Directions in sociolinguistics: Ethnography of communication* (pp. 407-434). New York, NY: Holt, Rinehart, & Winston.

Blommaert, J. (Ed.). (1999). *Language Ideological Debates.* Berlin: Mouton de Gruyter.

Bloom, B.S., et al. (1956). *Taxonomy of educational objectives: The classification of educational goals.* New York: David McKay Company.

Bloom, L. M. (1970). Language development: Form and function in emerging grammars. Cambridge: MIT Press.

Blum-Kulka, S. (2017). Language Socialization and Family Dinnertime Discourse. In Duff, P.A. & May, S. (Eds.), Language Socialization, 3rd Edition. New York, NY: Springer Publishing. 65-78.

Blum-Kulka, S., & Olshtain, E. (1984). Requests and apologies: A cross-cultural study of speech act realization patterns. *Applied Linguistics, 5*(3), 196-212.

Bolden, G.B. (2009). Implementing incipient actions: The discourse marker 'so' in English conversation. *Journal of Pragmatics,* 41. 974-998.

Boudourides, M. A. (2003). Constructivism, Education, Science, and Technology. *Canadian Journal of Learning and Technology,* 29(3). Retrieved March 02, 2016, from http://www.cjlt.ca/index.php/cjlt/article/viewArticle/83/77

Bourdieu, P. (1977a). Outline of a theory of practice (translated by Richard Nice). Cambridge: Cambridge University Press.

Bourdieu, P. (1977b). The Economics of Linguistic Exchanges. *Social Science Information,* 16(6). 645-668.

Bourdieu, P. (1984). Distinction: A Social Critique of the Judgment of Taste. Cambridge, MA: Harvard University Press.

Bourdieu, P. (1990). The logic of practice. Palo Alto: Stanford University Press.

Bourdieu, P., Passeron, J-C., & de Saint Martin, M. (1994). Students and the language of teaching. In P. Bourdieu, J-C. Passeron, & M. de Saint Martin (Eds.), *Academic discourse: Linguistic misunderstanding and professorial power* (pp. 35–79). Palo Alto, CA: Stanford University Press

Bracey, G.W. (1993). From Normal to Nerd: And Back Again. *The Phi Delta Kappan,* 74(9). 731-733.

Brickhouse, N. W. (2001). Embodying science: A feminist perspective on learning. *Journal of Research in Science Teaching, 38*(3), 282–295.

Brickhouse, N. W., & Potter, J. T. (2001). Young women's scientific identity formation in an urban context. *Journal of Research in Science Teaching, 38*(8), 965–980.

Bronson, M. C., & Watson-Gegeo, K. A. (2008). The critical moment: Language socialization and the (re)visioning of first and second language learning. In P. A. Duff & N. H. Hornberger (Eds.), Encyclopedia of language and education: Vol. 8. Language socialization (pp. 43–55). New York: Springer.

Brown, R., Cazden, C., & Bellugi, U. (1968). The child's grammar from I to III. In C. N. Cofer & B. S. Musgrave (Eds.), *Verbal behavior and learning: Problems and processes* (pp. 158–197). New York: McGraw-Hill.

Brown, R, & Gilman, A. (1972). Pronouns of power and solidarity. In Pier Gigliogli (Ed.), *Language and Social Context.* Harmondsworth: Penguin.

Brown, P. & Levinson, S.C. (1978). Politeness: Some universals in language usage. *Studies in Interactional Sociolinguistics 4.* Cambridge: Cambridge University Press.

Brummernhenrich, B. & Jucks, R. (2016). "He shouldn't have put it that way!" How face threats and mitigation strategies affect person perception in online tutoring, *Communication Education,* 65:3, 290-306, DOI: 10.1080/03634523.2015.1070957.

Bucholtz, M. (2001) The Whiteness of Nerds: Superstandard English and Racial Markedness. *Journal of Linguistic Anthropology* 11(1): 84-100.

Bucholtz, M. (2007) Word up: Social meanings of slang in California youth culture. In L. Monaghan and J.E. Goodman (Eds.), *A Cultural Approach to Interpersonal Communication: Essential Readings*. 243–67. Malden, MA: Blackwell Publishing.

Bucholtz, M. & Hall, K. (2004). Language and Identity. In A. Duranti (Ed.), *A Companion to Linguistic Anthropology* (pp. 369-394). Malden, MA: Blackwell Publishing.

Bunch, G. C. (2009). "Going up there": Challenges and opportunities for language minority students during a mainstream classroom speech event. *Linguistics and Education, 20*, 81–108.

Burke, P. J., & Stets, J. E. (2009). *Identity theory*. New York, NY: Oxford University Press.

Butler, J. (1990). *Gender trouble: Feminism and the subversion of identity.* New York, NY: Routledge.

Campos, J. & Sternberg, C. (1981). Perception, appraisal, and emotion: The onset of social referencing. In M.E. Lamb & L.R. Sherrod (Eds.), *Infant social cognition.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Carlone, H. B., & Johnson, A. (2007). Understanding the science experiences of successful women of color: Science identity as an analytic lens. *Journal of Research in Science Teaching, 44*(8), 1187–1218.

Cassell, J. (2002). Genderizing HCI. In J. Jacko & A. Sears (Eds.), *The handbook of human computer interaction.* Mahwah, NJ: Lawrence Erlbaum. 402-411.

Carr, P. B., & Steele, C. M. (2009). Stereotype threat and inflexible perseverance in problem solving. *Journal of Experimental Social Psychology,* doi:10.1016/j.jesp.2009.03.003

Carter, L. (2006). *Why Students with an Apparent Aptitude for Computer Science Don't Choose to Major in Computer Science.* In Proceedings of the 37th SIGCSE technical symposium on Computer science education, 27-31.

Cech, E. A. (2014). Culture of disengagement in engineering education? *Science, Technology & Human Values, 39*(1), 42–72.

Chafe, W., & Nichols, J. (1986). *Evidentiality: The linguistic coding of epistemology.* Norwood, NJ: Ablex. Cheryan, S., et al., (2009). Ambient belonging: How stereotypical cues impact gender participation in computer science. *Journal of Personality and Social Psychology,* 87. 1045-1060.

Cheryan, S., Meltzoff, A. N., & Kim, S. (2011). Classrooms matter: The design of virtual classrooms influences gender disparities in computer science classes. *Computers & Education,* 57, 1825-1835.

Cheryan, S., et al., (2011). Do female and male role models who embody STEM stereotypes hinder women's anticipated success in STEM? *Social Psychological and Personality Science,* 2. 656-664.

Clausen, J. A. (1968). *Socialization and Society*. Boston, MA: Little Brown and Company.

Clear, A. et al., (2017). CC2020: A Vision on Computing Curricula. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education.* 647-648.

Clear, T. (2006). Valuing computer science education research? *Proceedings of the 6th Baltic Sea conference on Computing education research.* 8-18.

Clegg, S. (2001). Theorising the Machine: Gender, Education and Computing. *Gender and Education,* 74(2). 307-324.

Cliburn, D.C. (2003) Experiences with Pair Programming at a Small College. *Computing Sciences in Colleges,* 19(1). 20-29.

Cockburn, A., & Williams, L. (2000). The Costs and Benefits of Pair Programming. *Proceedings of the First International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2000).*

Cohn, C. (1987). Sex and death in the rational world of defense intellectuals. *Signs: Journal of Women in Culture and Society, 12*, 687–718.

Cohoon, J.M. & Lord, H. (2006). A faculty role in women's participation in computing. In E.M. Trauth (Ed.), *Encyclopedia of gender and information technology.* Hershey, PA: Idea Publishing. 297-303.

Cole, M. (1985). The zone of proximal development: Where culture and cognition create each other. In J. Wertsch (Ed.), *Culture, communication, and cognition: Vygotskian perspectives* (pp. 146-161). Cambridge, UK: Cambridge University Press.

Cole, M., & Cole, S. (1989). *The development of children.* New York: Scientific American Books.

Connell, R. W. *Masculinities.* (1990). 2nd ed. Berkeley: U of California.

Cook, H.M. (1999). Language socialization in Japanese elementary schools: Attentive listening and reaction turns. *Journal of Pragmatics, 31*, 1443–1465.

Cook, H.M. & Burdelski, M. (2017). Language Socialization in Japanese Communities. In Duff, P.A. & May, S. (Eds.), Language Socialization, 3rd Edition. New York, NY: Springer Publishing. 309-322.

Cundiff, J.L., et al. (2013) Do gender-science stereotypes predict science identification and science career aspirations among undergraduate science majors? *Social Psychology of Education,* 16. 541-554.

Daly, N., Holmes, J., Newton, J., & Stubbe, M. (2004). Expletives as solidarity signals in FTAs on the factory floor. Journal of Pragmatics, 36, 945–964.

Davies, B. and Harré, R. (1990). Positioning: The Discursive Production of Selves. *Journal for the Theory of Social Behaviour, 20*(1), 44-63.

Dasgupta, N., et al. (2015) Female peers in small work groups enhance women's motivation, verbal participation, and career aspirations in engineering. *Proceedings of the National Academy of Sciences,* 112: 4988-4993.

Dempsey, J., Snodgrass, R. T., Kishi, I., & Titcomb, A. (2015). The emerging role of self-perception in student intentions. In *Proceedings of the 46th ACM technical symposium on computer science education* (pp. 108–113). New York, NY: ACM.

Dennehy, T.C. & Dasgupta, N. (2017). Female peer mentors early in college increase women's positive academic experiences and retention in engineering. *Proceedings of the National Academy of Sciences of the United States of America,* 114(23): 5964-5969.

Di Vesta, F. J. (1987). The Cognitive Movement and Education. In J. A. Glover & R. R. Ronning (Eds.), *Historical foundations of educational psychology* (203 - 233). New York, Plenum

Doubé, W. & Lang, C. (2012). Gender and stereotypes in motivation to study computer programming for careers in multimedia. *Computer Science Education,* 22(1). 63-78.

Downey, A. (2015). *Think Python: How to Think Like a Computer Scientist.* Needham, MA: Green Tea Press.

Downey, G., & Lucena, J. (2003). When students resist: Ethnography of a senior design experience in engineering education. *International Journal of Engineering Education, 19*(1), 168–176.

Drew, P. (2017). The Interface between Pragmatics and Conversation Analysis. In Ilie, C. & Norrick, N. (Eds.), *Pragmatics and its Interfaces.* Amsterdam, The Netherlands: John Benjamins. 59-84.

Du Bois, J.W. (2007). The stance triangle. In R. Englebretson (Ed.), *Stance in Discourse: Subjectivity in Interaction*. 13–182. Amsterdam, The Netherlands: Benjamins.

Dubois, B.L., & Crouch, I. (1975). The question of tag questions in women's speech: They don't really use more of them, do they? *Language in Society,* 4(3). 289-294.

Duff, P. A. (1995). An ethnography of communication in immersion classrooms in Hungary. TESOL Quarterly, 29, 505–537.

Duff, P. A. (1996). Different languages, different practices: Socialization of discourse competence in dual-language school classrooms in Hungary. In K. Bailey & D. Nunan (Eds.), Voices from the language classroom: Qualitative research in second language acquisition (pp. 407–433). New York: Cambridge University Press.

Duff, P. A. (2002). The discursive construction of knowledge, identity, and difference: An ethnography of communication in the high school mainstream. Applied Linguistics, 23, 289–322.

Duff, P. A. (2003). New directions in second language socialization research. Korean Journal of English Language and Linguistics, 3, 309–339.

Duff, P. A. (2004). Intertextuality and hybrid discourses: The infusion of pop culture in educational discourse. Linguistics and Education, 14, 231–176.

Duff, P. A. (2007a). Second language socialization as sociocultural theory: Insights and issues. Language Teaching, 40, 309–319.

Duff, P. A. (2007b). Problematising academic discourse socialisation. In H. Marriott, T. Moore, & R. Spence-Brown (Eds.), Learning discourses and the discourses of learning (pp. 1–18). Melbourne, Australia: Monash University e-Press/University of Sydney Press.

Duff, P.A. (2008a). Language Socialization, Participation and Identity: Ethnographic Approaches. In M. Martin-Jones, A. M. de Mejia and N. H. Hornberger (Eds),

Encyclopedia of Language and Education, 2nd Edition, Volume 3: Discourse and Education, 107–119.

Duff, P.A. (2008b). Language Socialization, Higher Education, and Work. In P.A. Duff & N. H. Hornberger (Eds), *Encyclopedia of Language and Education, 2nd Edition, Volume 8: Language Socialization*, 257-270.

Duff, P. A. (2009). Language socialization in a Canadian secondary school: Talking about current events. In R. Barnard & M. Torres-Guzman (Eds.), Creating communities of learning in schools (pp. 165–185). Clevedon, UK: Multilingual Matters.

Duff, P.A. (2010). Language Socialization into Academic Discourse Communities. *Annual Review of Applied Linguistics, 30,* 169-192.

Duff, P.A. (2011). Second language socialization. In A. Duranti, E. Ochs, & B. Schieffelin (Eds.), The handbook of language socialization (pp. 564–585). Oxford: Wiley-Blackwell.

Duff, P.A. (2017). Language Socialization, Higher Education, and Work. In Duff, P.A. & May, S. (Eds.), Language Socialization, 3rd Edition. New York, NY: Springer Publishing. 255-272.

Duff, P. A. & Kobayashi, M. (2010). The intersection of social, cognitive, and cultural processes in language learning. In R. Batstone (Ed.), Sociocognitive perspectives on language use and language learning (pp. 75–93). Oxford, UK: Oxford University Press.

Duff, P.A. and Labrie, N., (Guest eds.): 2000, 'Languages and work', Canadian Modern Language Review 57, 1, Special issue.

Duff, P.A., Wong, P. and Early, M. (2000). Learning language for work and life: The linguistic socialization of immigrant Canadians seeking careers in healthcare. Canadian Modern Language Review 57, 9–57.

Duffy, T. M., & Jonassen, D. H. (Eds.). (1992). *Constructivism and the technology of instruction: A conversation.* New York, NY: Routledge. [Kindle DX version]. Retrieved from Amazon.com

Duke, R. (2017). MLC and Language Socializations on StackOverflow: A Community of Practice Approach. Master's Thesis. University of California, Davis.

Dunn, C.D. (1999). Coming of age in Japan: language ideology and the acquisition of formal speech registers. In J. Verschueren (Ed.), *Language and Ideology: Selected Papers from the Sixth International Pragmatics Conference, 1,* 89-97. Antwerp: International Pragmatics Association.

Duranti, A. (1981). *The Samoan fono: A sociolinguistic study* (Pacific Linguistics Monographs, Series B, Vol. 80). Canberra, Australia: Australian National University, Department of Linguistics.

Duranti, A. (1985). Sociocultural dimensions of discourse. In T. A. V. Dijk (Ed.), Handbook of discourse analysis, Disciplines of discourse (Vol. 1, pp. 193–230). New York: Academic.

Duranti, A. (1994). *From Grammar to Politics: Linguistic Anthropology in a Western Samoan Village.* Berkeley, CA: University of California Press.

Eccles, J.S. et al. (1999). Linking gender to educational, occupational, and recreational choices: Applying the Eccles et al. model of achievement related choices. In W.B. Swann, Jr. & J.H. Langlois (Eds.), *Sexism and stereotypes in modern society: The gender science of Janet Taylor Spence.* Washington: American Psychological Association. 153-192.

Eckert, P. & McConnell-Ginet, S. (1992). Think Practically and Look Locally: Language and gender as community-based practice. *Annual Review of Anthropology, 21*. 561-490.

Eckert, P. & McConnell-Ginet, S. (1995) Constructing Meaning, Constructing Selves: Snapshots of Language, Gender, and Class from Belten High. In K. Hall & M. Bucholtz (Eds.), *Gender Articulated: Language and the Socially Constructed Self* (pp. 469-507). New York, NY: Routledge.

Eckert, P, and McConnell-Ginet, S. (2013). *Language and Gender.* Cambridge, UK: Cambridge University Press.

Eisenhart, M. A., & Finkel, E. (1998). *Women's science: Learning and succeeding from the margins*. Chicago, IL: University of Chicago Press.

English, L. D. (2003). Reconciling Theory, Research, And Practice: A models and modelling perspective. *Educational Studies in Mathematics* 54(2 & 3), 225 – 248.

Ensmenger, N. (2010). Making programming masculine. *Gendered codes: Why women are leaving computing* (pp. 115–142). Hoboken, NJ: Wiley.

Ervin-Tripp, S., & Gordon, D. (1984). The development of requests. In R. Sceifelbusch (Ed.), *Communicative competence: Assessment and intervention.* Baltimore: University Park Press.

Evaldsson, A.-C. (2002) Boys' gossip telling: Staging identities and indexing (unacceptable) masculine behavior. 22(2): 199–225.

Fader, A.: 2001, 'Literacy, bilingualism, and gender in a Hasidic community', Linguistics and Education 12(3), 261–283.

Faulkner, W. (2001). The technology question in feminism: A view from feminist technology studies. *Women's Studies International Forum, 24*(1), 79–95.10.1016/S0277-5395(00)00166-7

Field, M.: 2001, 'Triadic directives in Navajo language socialization', Language in Society 30(2), 249–263.

Fisher, A., Margolis, J., & Miller, F. (1997). Undergraduate Women in Computer Science: Experience, Motivation and Culture. *SIGCSE '97:* 106-110.

Fillmore, C.J. (1979) On Fluency. In C.J. Fillmore, D. Kempler & W. S-Y Wang (Eds.), *Individual differences in language ability and language behavior.* London: Academic Press. 85-102.

Flowers, J.C. (2018). How Is It Okay to Be a Black Nerd? In K.E. Lane (Ed.), *Age of the Geek.* London, UK: Palgrave Macmillan. 169-191.

Fraser, B. (1996). Pragmatic Markers. *Pragmatics,* 6(2). 167-190.

Fraser, B. (2011). The Sequencing of Contrastive Discourse Markers in English. *Baltic Journal of English Language, Literature and Culture.* 1, 29-35

Frisby, B.N., et al. (2014). Participation Apprehensive Students: The Influence of Face Support and Instructor-Student Rapport on Classroom Participation. *Communication Education.* 63(2), 105-123.

Fuller, J.M. (2003). Discourse marker use across speech contexts: A comparison of native and non-native speaker performance. *Multilingua,* 22. 185-208.

Garrett, P.B. & Baquedano-Lopez, P. (2002). Language Socialization: Reproduction and Continuity, Transformation and Change. *Annual Review of Anthropology, 31,* 339-361.

Gee, J. P. (2001). Identity as an analytic lens for research in education. *Review of Research in Education, 25*, 99–125.

Gee, J.P. (2014). *How to do Discourse Analysis: A Toolkit.* 2nd Edition. New York, NY: Routledge.

Gershuny, J. (2003). Web Sue and Net Nerds: A Neofuncionalist Analysis of the impact of Information Technology in the Home. *Social Forces,* 82(1). 141-168.

Gibbs, R. (1986). What makes some indirect speech acts conventional? *Journal of Memory and Language.* 25, 191-196.

Givon, T. (1989). *Mind, code, and context: Essays in pragmatics.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Goffman, E. (1967). On Face-Work: An analysis of Ritual Elements in Social Interaction. In *Interaction Ritual: Essays on Face-to-Face Behaviour.* London, UK: Penguin Books Ltd.

Gombert, J.E. (1992). *Metalinguistic Development.* Chicago, IL: University of Chicago Press.

Good, C., et al. (2008). Problems in the pipeline: Stereotype threat and women's achievement in high-level math courses. *Journal of Applied Developmental Psychology,* 29. 17-28.

Goodenough, W.H. (1965) Rethinking 'status' and 'role': Toward a general model of the cultural organization of social relationships. In M. Banton (Ed.), *The Relevance of Models for Social Anthropology*. 1–24. London: Tavistock.

Goodenow, C. (1993). Classroom belonging among early adolescent students: Relationships to motivation and achievement. *Journal of Early Adolescence, 13*, 21–43.

Goodwin, C. (1979). The interactive construction of a sentence in natural conversation. In G. Psathas (Ed.), *Everyday language: Studies in ethnomethodology* (pp. 97-121). New York, NY: Irvington.

Goodwin, C. (1994). Professional vision. *American Anthropologist, 96,* 606-633.

Goodwin, C. & Goodwin, M.H. (1996). Formulating Planes: Seeing as a Situated Activity. In D. Middleton & Y. Engestrom (Eds.), *Cognition and Communication at Work* (pp. 61-95). Cambridge, UK: Cambridge University Press.

Goodwin, C. & Heritage, J. (1990). Conversational analysis. *Annual Review of Anthropology, 19,* 283-307.

Goodwin, M.H. (1990). *He-said-she-said: Talk as social organization among Black children.* Bloomington, IN: Indiana University Press.

Goodwin, M. H., & Kyratzis, A. (2012). Peer language socialization. In A. Duranti, E. Ochs, & B. B. Schieffelin (Eds.), The handbook of language socialization (pp. 365–390). Malden: Wiley-Blackwell.

Goodwin, M.H. (1990) He - Said - She - Said: Talk as Social Organization Among Black Children. Bloomington, IN: Indiana University Press.

Goodwin, M.H. (2006) The Hidden Life of Girls: Games of Stance, Status, and Exclusion. Oxford: Blackwell.

Goodwin, C. & Goodwin, M.H. (1987) Concurrent operations on talk: Notes on the interactive organization of assessments. IPrA Papers in Pragmatics 1 (1): 1 – 52.

Gordon, D. (2004). "I'm Tired. You Clean and Cook.": Shifting Gender Identities and Second Language Socialization. *TESOL Quarterly, 36*(3), 437-457.

Gordon, D. & Lakoff, G. (1975). Conversational postulates. In P. Cole & J. Morgan (Eds.), *Syntax and semantics* (Vol. 3, pp. 83-106). New York, NY: Academic.

Grice, H.P. (1975). Logic and Conversation. In Cole, P. & Morgan, J. (Eds.), *Syntax and Semantics.* New York: Academic Press.

Guardado, M. (2009). Speaking Spanish Like a Boy Scout: Language Socialization, Resistance, and Reproduction in a Heritage Language Scout Troop. *The Canadian Modern Language Review, 66*(1), 101-129.

Gumperz, J. J. (1968). The speech community. In D. L. Sils & R. K. Merton (Eds.), International encyclopedia of the social sciences (pp. 381–386). New York: Macmillan.

Gumperz, J.J. (1982a). *Discourse strategies.* Cambridge, UK: Cambridge University Press.

Gumperz, J.J. (Ed.). (1982b). *Language and social identity.* Cambridge, UK: Cambridge University Press.

Gumperz, J. J., & Hymes, D. (Eds.). (1972). Directions in sociolinguistics: The ethnography of communication. New York: Holt, Rinehart and Winston.

Gunnarsson, B. (2009). *Professional discourse*. London, UK: Continuum.

Halliday, M.A.K. (1975). *Learning how to mean.* London, UK: Arnold.

Hancock, D. (2004). Cooperative Learning and Peer Orientation Effects on Motivation and Achievement. *The Journal of Education Research,* 97(3), 159-166.

Hanks, B., et al. (2004). Program Quality with Pair Programming in CSI. *ACM SIGCSE,* 36. 176-180.

Hansen, A.K., Dwyer, H.A., Iveland, A., Talesfore, M., Wright, L., Harlow, D. B., & Franklin, D. (2017). Assessing children's understanding of the work of computer scientists: The draw-a-computer-scientist test. In *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education* (pp. 279–284). New York, NY: ACM.

Hapnes, T. & Rasmussen, B. (1998). Excluding Women from the Technologies of the Future? A Case Study of the Culture of Computer Science. In Hopkins, P (Ed.) *Sex/machine: Readings in Culture, Gender and Technology.* Bloomington and Indianapolis: Indiana University Press. 381-394.

He, A.W. (2001). The language of ambiguity: practices in Chinese heritage language classes. *Discourse Studies, 3*(1), 75-96.

Heath, S. B. (1983). Ways with words: Language, life, and work in communities and classrooms. Cambridge, UK: Cambridge University Press.

Heath, S.B. (1989). Oral and literate traditions among Black Americans living in poverty. *American Psychologist, 44*(2), 367–373.

Heath, S.B. (2017). Language Socialization in the Learning Communities of Adolescents. In Duff, P.A. & May, S. (Eds.), Language Socialization, 3rd Edition. New York, NY: Springer Publishing. 213-226.

Heller, M. (1998). Strategic ambiguity: Codeswitching in the management of conversation. In M. Heller (Ed.), *Codeswitching: Anthropological and sociolinguistic perspectives* (pp. 77-96). Berlin: Mouton de Gruyter.

Heller, M. (2002). Globalization and the commodification of bilingualism in Canada. In D. Block and D. Cameron (Eds.), *Globalization and Language Teaching* (pp. 47-63). London, UK: Routledge.

Heritage, J. (1984a). A change-of-state token and aspects of its sequential placement. In J.M. Atkinson & J. Heritage (Eds.), *Structures of social action: Studies in conversation analysis* (pp. 299-345). Cambridge, UK: Cambridge University Press.

Heritage, J. (1984b). *Garfinkel and ethnomethodology.* Cambridge, UK: Polity.

Herring, C. (2009). Does diversity pay?: Race, gender, and the business case for diversity. *American Sociological Review,* 74. 208-224.

Hiebert, J. et al. (1996). Problem Solving as a Basis for Reform in Curriculum and Instruction: The Case of Mathematics. *Educational Researcher, 25*(12), 12-21.

Hobbs, P. (2004). The role of progress notes in the professional socialization of medical residents. Journal of Pragmatics, 36, 1579–1607.

Holland, J. L. (1997). *Making vocational choices: A theory of vocational personalities and work environments* (3rd ed.). Odessa, FL: Psychological Assessment Resources.

Holmes, J. (2005a). Story-telling at work: A complex discursive resource for integrating personal, professional, and social identities. Discourse Studies, 7, 671–700.

Holmes, J. (2005b). When small talk is a big deal: Sociolinguistic challenges in the workplace. In M. H. Long (Ed.), Second language needs analysis (pp. 344–372). Cambridge: Cambridge University Press.

Holmes, J., & Schnurr, S. (2005). Politeness, humor and gender in the workplace: Negotiating norms and identifying contestation. Journal of Politeness Research, 1, 121–149.

Holmes, J., & Stubbe, M. (2003). Power and politeness in the workplace: A sociolinguistic analysis of talk at work. London: Pearson Education.

Howard, K. M., & Lo, A. (Eds.). (2009). Mobilizing respect and politeness in classrooms [Special issue]. Linguistics and Education, 20(3).

Huff, C. (2002). Gender, software design, and occupational equity. *ACM SIGCSE Bulletin, 24,* 112-115.

Hull, G. (Ed.). (1997). *Changing Work, Changing Workers: Critical Perspectives on Language, Literacy, and Skills*. Albany, NY: State University of New York Press.

Hur, J.W. et al. (2017). Girls and computer science: experiences, perceptions, and career aspirations. *Computer Science Education,* 27(2). 100-120.

Hyland, K. (2006). English for academic purposes. New York: Routledge.

Hymes, D. (1964). Introduction: Toward ethnographies of communication. American Anthropologist, 66(6), Part 2:1–34.

Hymes, D. (1972a). On communicative competence. In J. B. Pride & J. Holmes (Eds.), Sociolinguistics (pp. 269–285). Harmondsworth: Penguin.

Hymes, D. (1972b). Models of the interaction of language and social life. In J. J. Gumperz & D. Hymes (Eds.), Directions in sociolinguistics: The ethnography of communication (pp. 35–71). New York: Holt, Rinehart and Winston.

Hymes, D. (1975). Breakthrough into Performance. In D. Ben-Amos & K.S. Goldstein (Eds.), *Folklore: Performance and Communication* (pp. 11-74). The Hague: Mouton.

Iedema, R. (2003). Discourses of post-bureaucratic organisation. Amsterdam: John Benjamins.

Inzlicht, M., & Ben-Zeev, T. (2000). A Threatening Intellectual Environment: Why Females Are Susceptible to Experiencing Problem-Solving Deficits in the Presence of Males. *Psychological Science,* 11(5), 365-371.

Irvine, J.T. (1990). Registering affect: Heteroglossia in the linguistic expression of emotion. In C. Lutz & L. Abu-Lughod (Eds.), *Language and the politics of emotion* (pp. 126-185). New York, NY: Cambridge University Press.

Irvine, J.T. (2001). "Style" as Distinctiveness: The Culture and Ideology of Linguistic Differentiation. In P. Eckert & J.R. Rickford (Eds.), *Style and Sociolinguistic Variation* (pp. 21-43). Cambridge, UK: Cambridge University Press.

Irvine, J.T. & Gal, S. (2000). Language Ideology and Linguistic Differentiation. In P.V. Kroskrity (Ed.), *Regimes of Language: Ideologies, Polities, and Identities* (pp. 35-84). Santa Fe, NM: School of American Research Press.

Israel, M. et al. (2016). Assessing collaborative computing: development of the Collaborative-Computing Observation Instrument (C-COI). *Computer Science Education,* 26(2-3). 208-233.

Jack, G. (2009). A critical perspective on teaching intercultural competence in a management department. In A. Feng, M. Byram, & M. Fleming (Eds.), Becoming interculturally competent through education and training (pp. 95–114). Bristol, UK: Multilingual Matters.

Jacobs-Huey, L. (1999). Becoming cosmetologists: language socialization and identity in an African American beauty college. PhD thesis. Univ. Calif., Los Angeles.

Jacobs-Huey, L. (2003). Ladies are seen, not heard: Language socialization in a southern, African American cosmetology school. Anthropology and Education Quarterly, 34(3), 277–299.

Jacoby, S. (1998) *Science as Performance: Socializing Scientific Discourse Through the Conference Talk Rehearsal.* PhD dissertation, University of California, Los Angeles.

Jacoby, S. and Gonzales, P.: 1991, 'The constitution of expert-novice in scientific discourse', Issues in Applied Linguistics 2, 149–181.

Jacoby, S. & Ochs, E. (1995) Co-Construction: An Introduction. *Research on Language and Social Interaction, 28*(3), 171-183.

Johns, A. (2005). English for academic purposes: Issues in undergraduate reading and writing. In P. Bruthiaux, D. Atkinson, W. Eggington, W. Grabe, & V. Ramanathan (Eds.), Directions in applied linguistics (pp. 101–116). Clevedon, UK: Multilingual Matters.

Jupp, T., Roberts, C., & Cook-Gumperz, J. (1982). Language and disadvantage: The hidden process. In J. Gumperz (Ed.), Language and social identity (pp. 232–256). Cambridge, MA: Cambridge University Press.

Kaendler, C. et al. (2014). Teacher Competencies for the Implementation of Collaborative Learning in the Classroom: a Framework and Research Review. *Educational Psychology Review,* 27(3). 505-536.

Kanny, M.A., Sax, L.J., & Riggers-Piehl, T.A. (2014). Investigating forty years of STEM research: How explanations for the gender gap have evolved over time. *Journal of Women and Minorities in Science and Engineering, 20*(2), 127–148.

Katz, M.L. (2000). Workplace language teaching and the intercultural construction of ideologies of competence. *Canadian Modern Language Review, 57*, 144–172.

Katz, S., et al. (2006). Gender, Achievement, and Persistence in an Undergraduate Computer Science Program. *The DATA BASE for Advances in Information Systems,* 37(4), 42-57.

Kendall, L. (2000). Oh No! I'm a Nerd: Hegemonic Masculinity on an Online Forum. *Gender and Society,* 14(2). 256-274.

Kerssen-Griep, J. (2001). Teacher Communication Activities Relevant to Student Motivation: Classroom Facework and Instructional Communication Competence. *Communication Education.* 50(3). 256-273.

Kerssen-Griep, J., Hess, J.A., & Trees, A.R. (2003). Sustaining the desire to learn: Dimensions of perceived instructional facework related to student involvement and motivation to learn. *Western Journal of Communication.* 67(4). 357-381.

Kerssen-Griep, J., Trees, A.R., & Hess, J.A. (2008). Attentive Facework During Instructional Feedback: Key to Perceiving Mentorship and an Optimal Learning Environment. *Communication Education.* 57(3). 312-332. DOI: 10.1080/10570310309374779.

Kim, J. & Duff, P.A. (2012). The Language Socialization and Identity Negotiations of Generation 1.5 Korean-Canadian University Students. *TESL Canada Journal, 29*(6), 81-102.

Kinnunen, P. et al., (2013). Getting to know computer science freshmen. *Proceedings of the 13th Koli Calling International Conference on Computing Educational Research.* Koli, Finland: ACM. 59-66.

Klopp, M., et al. (2017). Can Pair Programming Address Multidimensional Issues in Higher Education? *International Conference on Interactive Collaborative Learning.* 479-486.

Kobayashi, M. (2003). The role of peer support in students' accomplishment of oral academic tasks. *Canadian Modern Language Review, 59,* 337–368.

Kobayashi, M. (2006). Second language socialization through an oral project presentation: Japanese university students' experience. In G. H. Beckett & P. C. Miller (Eds.), *Project based second and foreign language education* (pp. 71–93). Charlotte, NC: Information Age.

Kövecses, Z. and G. Radden. (1998). Metonymy: Developing a Cognitive Linguistic View. *Cognitive Linguistics*, 9. 37-77.

Kroskrity, P.V. (Ed.). (2000). *Regimes of Language: Ideologies, Polities, and Identities.* Santa Fe, NM: School of American Research Press.

Kulick, D., & Schieffelin, B. B. (2004). Language socialization. In A. Duranti (Ed.), A companion to linguistic anthropology (pp. 349–368). Oxford: Blackwell.

Lachney, M. (2018). Computational communities: African-American cultural capital in computer science education. *Computer Science Education.* 27(3-4). 175-196.

Lagesen, V.A. (2007). The Strength of Numbers: Strategies to Include Women into Computer Science. *Social Studies of Science,* 37(1). 67092.

Lakoff, G. (1972). Hedges: A study in meaning criteria and the logic of fuzzy concepts. *Papers from the Eighth Regional Meeting of the Chicago Linguistics Society* (pp. 271-291). Chicago, IL: University of Chicago.

Lakoff, R.T. (1973). The logic of politeness, or minding your P's and Q's. *Papers from the Ninth Regional Meeting of the Chicago Linguistic Society* (pp. 292-305). Chicago, IL: University of Chicago.

Lakoff, R.T. (1979). Stylistic Strategies Within A Grammar Of Style. *Annals of the New York Academy of Sciences,* 327 (In *Language, Sex, and Gender: Does La Difference Make a Difference?*), 51-51. DOI: 10.1111/j.1749-6632.1979.tb17752.

Lakoff, R.T. (2005). Introduction: Broadening the horizon of linguistic politeness. In Lakoff, R.T. & Ide, S. (Eds.), *Broadening the Horizon of Linguistic Politeness.* Amsterdam: John Benjamins Publishing Co.

Lave, J. (1988). *Cognition in Practice: mind, mathematics and culture in everyday life.* Cambridge, MA: Cambridge University Press. [Kindle DX version]. Retrieved from Amazon.com

Lave, J. (1993). The practice of learning. In S. Chaiklin & J. Lave (Eds.), *Understanding practice: Perspectives on activity and context* (pp. 3-32). New York, NY: Cambridge University Press.

Lave, J. (1996). Teaching, as Learning, in Practice. *Mind, Culture, and Activity,* 3(3), 149-164

Lave, J. & Wenger, E. (1991a) Situated Learning: Legitimate Peripheral Participation. Cambridge, MA: Cambridge University Press.

Lave, J. & Wenger, E. (1991b). *Situated cognition: Legitimate peripheral participation.* New York, NY: Cambridge University Press.

Leder, G.C. & Vale, C.M. (2004). Student Views of Computer-Based Mathematics in the Middle Years: Does Gender Make a Difference? *Education Studies in Mathematics,* 56(2/3). 287-312.

Leichty, G. & Applegate, J.L. (1991). Social-cognitive and situational influences on the use of face-saving persuasive strategies. *Human Communication Research,* 17(3). 451-484.

Leech, G. (1983). *Principles of Pragmatics.* London: Longman.

Lehman, K.J. et al. (2016). Women planning to major in computer science: Who are they and what makes them unique? *Computer Science Education.* 26(4). 277-298.

Leontyev, A.N. (1981). *Situated Learning: Legitimate peripheral participation.* Cambridge, UK: Cambridge University Press.

Levinson, S. C. (1983). *Pragmatics.* Cambridge, UK: Cambridge University Press.

Lewis, C. M., Anderson, R. E., & Yasuhara, K. (2016, August). I don't code all day: Fitting in computer science when the stereotypes don't fit. In *Proceedings of the 2016 ACM conference on international computing education research* (pp. 23–32). New York, NY: ACM.

Lewis, C.M. & Shah, N. (2015) How equity and inequity can emerge in pair programming. *Proceedings of the eleventh annual international conference on international computing education research*. ACM. 41-50.

Lewis, C. M., Yasuhara, K., & Anderson, R. E. (2011). Deciding to major in computer science: A grounded theory of students' self-assessment of ability. In *Proceedings of the seventh*

international workshop on computing education research (pp. 3–10). New York, NY: ACM.

Lewis, Jr., N.A. & Sekaquaptewa, D. (2017). Beyond test performance: a broader view of stereotype threat. *Current Opinion in Psychology,* 11: 40-43.

Li, D. (2000). The pragmatics of making requests in the L2 workplace: A case study of language socialization. *Canadian Modern Language Review, 57*, 58–87.

Lim, T. & Bowers, J.W. (1991). Facework Solidarity, Approbation, and Tact. *Human Communication Research.* 17(3). 415-450. DOI: 10.1111/j.1468-2958.1991.tb00239.x

Lishinski, A. et al., (2016). Learning to Program: Gender differences and interactive effects of students' motivation, goals, and self-efficacy on performance. *Proceedings of the 12th Annual International ACM Conference on International Computing Education Research (ICER'16).* Melbourne, AU.

Lockard, C.B. & Wolf, M. (2012). Employment outlook: 2010-2020. Occupational employment projections to 2020. *Monthly Labor Review.* 84-108.

Lord, S. M., et al. (2009). Who's Persisting in Engineering? A comparative analysis of female and male Asian, Black, Hispanic, Native American, and White students. *Journal of Women and Minorities in Science and Engineering,* 15, 167-190.

Macionis, J. J. (2013). *Sociology* (15th ed.). Boston, MA: Pearson.

Magee, R.M. et al. (2011). *Gendering infrastructure, CHI 2011.* Vancouver, Canada.

Mak, B. C. N., Liu, Y. Q., & Deneen, C. C. (2012). Humor in the workplace: A regulating and coping mechanism in socialization. Discourse and Communication, 6(2), 163–179.

Mannix, E. & Neale, M.A. (2005). What differences make a difference? The promise and reality of diverse teams in organizations. *Psychological Science in the Public Interest,* 6. 32-55.

Margolis, J. & Fisher, A. (2002). *Unlocking the Clubhouse: Women in Computing.* The MIT Press: Cambridge, MA.

Matsumura, S. (2001). Learning the Rules for Offering Advice: A Quantitative Approach to Second Language Socialization. *Language Learning, 51*(4), 635-679.

Matthewman, J. (1996). Trends and developments in the use of competency frameworks. Competency, 4, 2–11.

Mawer, G. (1999). Language and literacy in workplace education: Learning at work. London: Longman.

McDowell, C. et al. (2002). The Effects of Pair-Programming on Performance in an Introductory Programming Course. *Presented at the 33rd ACM Technical Symposium on Computer Science Education.*

McGroarty, M. (1998). Constructive and Constructivist Challenges for Applied Linguistics. *Language Learning, 48*(4), 591-622.

Mehan, H. (1979). *Learning lessons*. Cambridge, MA: Harvard University Press.

Mendes, E., et al. (2005). Investigating Pair-Programming in a 2nd-Year Software Development and Design Computer Science Course. *ACM SIGCSE Bulletin, 37*, 296-300.

Mendes, E., et al. (2006). A Replicated Experiment of Pair Programming in a 2nd-Year Software Development and Design Computer Science Course. *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education.* 108-112.

Mendoza - Denton, N. (2008) Homegirls: Symbolic Practices in the Making of Latina Youth Styles. Oxford: Blackwell.

Mercier, E. et al. (2015). Researching and Designing for the Orchestration of Learning in the CSCL Classroom. *Proceedings of the 11th international conference on computer supported collaborative learning.* Gothenburg, Sweden: International Society of the Learning Sciences. 599-606.

Mertz, E. (1996). Recontextualization as socialization: Text and pragmatics in the law school classroom. In M. Silverstein and G. Urban (Eds.), *Natural Histories of Discourse* (pp. 229-249). Chicago, IL: The University of Chicago Press, Chicago.

Mertz, E. (1998). Linguistic ideology and praxis in US law school classrooms. In B. Schieffelin, K. Woolard, and P. Kroskrity (Eds.), *Language Ideologies: Practice and Theory*. Oxford, UK: Oxford University Press.

Mertz, E. (2007). *The Language of Law: Learning to Think Like a Lawyer*. Oxford, UK: Oxford University Press.

Miliszewska, I., Barker, G., Henderson, F., & Sztendur, F. (2006). The Issue of Gender Equity in Computer Science – What Students Say. *Journal of Information Technology Education,* 5, 107-120.

Miller, J.: 2003, Audible Difference: ESL and Social Identity in Schools, Multilingual Matters, Clevedon, UK.

Miller, P. (1986) Teasing as language socialization and verbal play in a white working class community. In B.B. Schieffelin & E. Ochs, (Eds.) *Language Socialization Across Cultures* (pp. 199-212). Cambridge, UK: Cambridge University Press.

Molle, D., & Prior, P. (2008). Multimodal genre systems in EAP writing pedagogy: Reflecting on a needs analysis. TESOL Quarterly, 42, 541–566.

Moore, L. (2011). Language Socialization and Repetition. In A. Duranti, E. Ochs, & B. Schieffelin (Eds.), *The handbook of language socialization* (pp. 209-226). Oxford: Wiley-Blackwell.

Morgan, M. (2002) Language, Discourse, and Power in African American Culture. Cambridge: Cambridge University Press.

Morita, N. (2000). Discourse socialization through oral classroom activities in a TESL graduate program. TESOL Quarterly, 34, 279–310.

Morita, N. (2002). Negotiating participation in second language academic communities: A study of identity, agency, and transformation. Unpublished doctoral dissertation, University of British Columbia, Vancouver, Canada.

Morita, N. (2004). Negotiating participation and identity in second language academic communities. TESOL Quarterly, 38, 573–603.

Morita, N. (2009). Language, culture, gender, and academic socialization. Language and Education, 23, 443–460.

Morita, N., & Kobayashi, M. (2008). Academic discourse socialization in a second language. In P. A. Duff & N. H. Hornberger (Eds.), Encyclopedia of language and education: Vol. 8. Language socialization (pp. 243–256). New York: Springer.

Nagappan, N. et al., (2003). Improving the CSI Experience with Pair Programming. *ACM SIGCSE Bulletin,* 35. 359-362.

National Science Foundation. (2013). *Women, minorities, and persons with disabilities in science and engineering: 2007.* Arlington, VA: National Science Foundation.

National Science Foundation, National Center for Science and Engineering Statistics. (2017). *Women, minorities, and persons with disabilities in science and Engineering: 2017. Special report NSF 17-310*. Arlington, VA.

Nelson, K.L. et al. (2013). Gender Differences in Fear of Failure Amongst Engineering Students. *International Journal of Humanities and Social Science,* 3. 10-16.

Newman, R.S. (1994). Adaptive help-seeking: A strategy of self-regulated learning. In D.H. Schunk & B.T. Zimmerman (Eds.), *Self-regulation of learning and performance: Issues and educational applications.* Hillsdale, NJ: Erlbaum. 283-301.

Norton, B. (2000). *Identity and Language Learning*. Pearson, New York.

Ochs, E. (1990). Indexicality and socialization. In J.W. Stigler, R. Shweder, & G. Herdt (Eds.), *Cultural psychology: Essays on comparative human development* (pp. 287-308). Cambridge, UK: Cambridge University Press.

Ochs, E. (1991). Socialization through Language and Interaction: A Theoretical Introduction. *Issues in Applied Linguistics,* 2(2), 143-147.

Ochs, E. (1993). Constructing Social Identity: A Language Socialization Perspective, *Research on Language and Social Interaction*, 26:3. 287-306.

Ochs, E. (2000). Socialization. *Journal of Linguistic Anthropology, 9*(1), 230-233.

Ochs, E., & Schieffelin, B. (1989). Language has a heart. *Text,* 9. 7-25.

Ochs, E & Schieffelin, B.B. (1994). The Impact of Language Socialization on Grammatical Development. In P. Fletcher & B. MacWhinney (Eds.), *Handbook of Child Language* (pp. 73-94). New York, NY: Basil Blackwell.

Ochs, E. & Schieffelin, B.B. (2017) Language Socialization: An Historical Overview. In Duff, P.A. & May, S. (Eds.), Language Socialization, 3rd Edition. New York, NY: Springer Publishing. 3-16.

Ochs, E., Solomon, O., & Sterponi, L. (2005). Limitations and transformations of habitus in child directed communication. Discourse Studies, 7, 547–583.

Oldenziel, R. (2001) Man the Maker, Woman the Consumer: The Consumption Junction Revisited. In Creager, A., Lunbeck, E. & Schiebinger, L. (Eds.) *Feminism in Twentieth Century Science, Technology and Medicine.* Chicago, IL: University of Chicago Press. 28-148.

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

Othman, Z. (2010). The use of *okay*, *right*, and *yeah* in academic lectures by native speaker lecturers: Their 'anticipated' and 'real' meanings. *Discourse Studies,* 12(5) 665-681.

Parks, S. (2001). Moving from school to the workplace: Disciplinary innovation, border crossings, and the reshaping of a written genre. *Applied Linguistics, 22*, 405–438.

Parks, S. and Maguire, M. (1999). Coping with on-the-job writing skills in ESL: A constructivist-semiotic perspective. *Language Learning, 49*, 143–175.

Paugh, A. (2005). Learning about work at dinnertime: Language socialization in dual-earner American families. *Discourse & Society, 16*, 55–78.

Paugh, A. (2012a). Playing with languages: Children and change in a Caribbean village. New York: Berghahn.

Paugh, A. (2012b). Local theories of children rearing. In A. Duranti, E. Ochs, & B. B. Schieffelin (Eds.), The handbook of language socialization (pp. 150–168). Malden: Wiley-Blackwell.

Person, N.K. et al. (1995). Pragmatics and Pedagogy: Conversational Rules and Politeness Strategies May Inhibit Effective Tutoring. *Cognition and Instruction.* 13(2). 161-199.

Perret-Clermont, A.-N., Perret, J.-F., & Bell, N. (1991). The social construction of meaning and cognitive activity in elementary school children. In L. B. Resnick, J. M. Levine, & S. D. Teasley (Eds.), Perspectives on socially shared cognition (pp. 41–62). Washington, DC: American Psychological Association.

Peters, A. K. (2014). *The role of students' identity development in higher education in computing* (Doctoral dissertation). Uppsala: Uppsala universitet.

Peters, A. K., Berglund, A., Eckerdal, A., & Pears, A. (2014). First year computer science and IT students' experience of participation in the discipline. In *2014 international conference on teaching and learning in computing and engineering (LaTiCE)* (pp. 1–8). Kuching: IEEE.

Peters, A. K., & Pears, A. (2012). Students' experiences and attitudes towards learning computer science. In 2012 *Frontiers in education conference (FIE)* (pp. 1–6). Seattle, WA: IEEE.

Peters, A. K., & Pears, A. (2013). Engagement in Computer Science and IT–What! A Matter of Identity? In *2013 international conference on teaching and learning in computing and engineering (LaTiCE)* (pp. 114–121). IEEE.

Phillips, R. (2005). Challenging The Primacy Of Lectures: The Dissonance Between Theory And Practice In University Teaching. *Journal of University Teaching & Learning Practice,* 2(1).

Philips, S. U. (1982). The language socialization of lawyers: Acquiring the "cant". In G. Spindler (Ed.), Doing the ethnography of schooling (pp. 176–209). New York: Holt, Rinehart and Winston.

Philips, S.U. (1983). *The Invisible Culture: Communication in Classroom and Community on the Warm Springs Indian Reservation,* Longman, New York.

Piaget, J. (1952). *The origins of intelligence in children.* New York, NY: International Universities Press.

Poncini, G. (2003). Multicultural business meetings and the role of languages other than English. Journal of Intercultural Studies, 24, 17–32.

Pozzer, L. L., & Jackson, P. A. (2015). Conceptualizing identity in science education research: Theoretical and methodological issues. In *Sociocultural studies and implications for science education* (pp. 213–230). Dordrecht: Springer Netherlands.

Resnick, L. B. (1991). Shared cognition: Thinking as social practice. In L. B. Resnick, J. M. Levine, & S. D. Teasley (Eds.), Perspectives on socially shared cognition (pp. 1–20). Washington, DC: American Psychological Association.

Riley, K.C. (2008). Language Socialization. In B. Spolsky & F.M. Hult, (Eds.), *The Handbook of Educational Linguistics* (pp. 399-410). Malden, MA: Blackwell Publishing.

Roberts, C. (2010). Language Socialization in the Workplace. *Annual Review of Applied Linguistics, 30,* 211-227.

Roberts, C., & Campbell, S. (2006). Talk on trial: Job interviews, language, and ethnicity. Retrieved from Department for Work & Pensions Web site: http://www.dwp.gov.uk/asd/asd5/rrs2006.asp#talkontrial.

Roberts, C., & Sarangi, S. (1999). Hybridity in gatekeeping discourse: Issues of practical relevance for the researcher. In S. Sarangi & C. Roberts (Eds.), Talk, work and institutional order (pp. 473–504). Berlin, Germany: Mouton de Gruyter.

Robertson, J. (2013). The influence of a game-making project on male and female learners' attitudes to computing. *Computer Science Education*, *23*(1), 58-83.

Robinson, D. (2017, September 6). *The Incredible Growth of Python*. Retrieved from http://www.stackoverflow.blog/2017/09/06/incredible-growth-python/

Rode, J.A. (2011) A theoretical agenda for feminist HCI. *Interacting with Computers,* 23. 393-400.

Rodriguez, F.J., et al. (2017). Exploring the Pair Programming Process: Characteristics of Effective Collaboration. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education.* 507-512.

Rodriguez, S.L. & Lehman, K. (2017). Developing the next generation of diverse computer scientists: the need for enhanced, intersectional computing identity theory. *Computer Science Education, 27*(3-4). 229-247.

Rogoff, B. (1990). Apprenticeship in thinking: Cognitive development in social context. New York: Oxford University Press.

Rogoff, B., et al. (Eds.). (2014). Learning by observing and pitching in to family and community endeavors. Human Development, 57(2–3), 150–161.

Rymes B. 2001. Conversational Borderlands: Language and Identity in an Urban Alternative High School. New York: Teachers Coll.

Sacks, H. (1972) On the analyzability of stories by children. In J. J. Gumperz and D. Hymes (Eds.), Directions in Sociolinguistics: The Ethnography of Communication. 325 – 45. New York: Holt, Rinehart and Winston.

Sacks, H., Schegloff, E.A., & Jefferson, G. (1974). A simplest systematics for the organization of turn-taking for conversation. *Language, 50,* 696-735.

Salleh, N., et al. (2010). Empirical Studies of Pair Programming for CS/SE Teaching in Higher Education: A Systematic Literature Review. *IEEE Transactions on Software Engineering,* 37(4). 509-525.

Salter, A. & Blodgett, B. (2017). *Toxic Geek Masculinity in Media.* London, UK: Palgrave Macmillan.

Sapir, E. (1924). Culture, Genuine and Spurious. *American Journal of Sociology,* 29(4).401-429.

Sapir, E. (1933). Language. *Encyclopaedia of the Social Sciences,* 9. 155-169.

Sarangi, S. and Roberts, C. (2002). Discoursal (mis)alignments in professional gatekeeping encounters. In C. Kramsch (Ed.), *Language Acquisition and Language Socialization* (pp. 187-227). New York, NY: Ecological Perspectives.

Sax, L. J., et al. (2017). Anatomy of an enduring gender gap: The evolution of women's participation in computer science. *The Journal of Higher Education, 88*(2), 258–293.

Schatzman, L. & Straus, A.L. (1973). *Field Research: Strategies for a natural sociology.* Englewood Cliffs, NJ: Prentice-Hall.

Schegloff, E.A. (1972). Sequencing in conversational openings. In J.J. Gumperz & D. Hymes (Eds.), *Directions in sociolinguistics: The ethnography of communication* (pp. 346-380). New York, NY: Hold, Rinehart & Winston.

Schegloff, E.A. (1982). Discourse as an interactional achievement: Some uses of "uh huh" and other things that come between sentences. In D. Tannen (Ed.), *Georgetown University Round Table on Languages and Linguistics* (pp. 71-93). Washington, DC: Georgetown University Press.

Schegloff, E.A. (1986). The routine as achievement. *Human Studies, 9,* 111-151.

Schegloff, E.A. & Sacks, H. (1973). Opening up closings. *Semiotica, 8,* 289-327.

Schieffelin, B.B., Woolard, K.A., & Kroskrity, P.V. (Eds.) (1998). *Language Ideologies: Practice and Theory.* New York, NY: Oxford University Press.

Schnurr, S. (2009). Constructing leader identities through teasing at work. Journal of Pragmatics, 41(6), 1125–1138.

Schulte, C., & Knobelsdorf, M. (2007). Attitudes towards computer science-computing experiences as a starting point and barrier to computer science. In *Proceedings of the third international workshop on computing education research* (pp. 27–38).

Sele, L. (2012). Talking Nerdy: The invisibility of female computer Nerds in Popular culture and the subsequent fewer number of women and girls in the computer sciences. *Journal of Integrated Studies,* 1(3), 1-14.

Searle, J.R. (1969). *Speech acts: An essay in the philosophy of language.* Cambridge, UK: Cambridge University Press.

Searle, J.R. (1975). Indirect speech acts. In P. Cole & J.L. Morgan (Eds.), *Syntax and semantics 3: Speech acts* (pp. 59-82). New York, NY: Academic Press.

Shaochun, X. & Rajlich, V. (2006). Empirical Validation of Test-Driven Pair Programming in Game Development. *Proceedings of the Fifth IEEE/ACIS International Conference on*

Computational and Information Science – in Conjunction with First IEEE/ACIS Workshop Component-Based Software Engineer Architecture and Reuse. 500-505.

Shashaani, L. (1994). Gender-Differences in Computer Experience and its Influence on Computer Attitudes. *Journal of Educational Computing Research,* 11(4), 347-367.

Shashaani, L. (1997). Gender Differences in Computer Attitudes and Use Among College Students. *Journal of Educational Computing Research,* 16(1), 37-51.

Sheldon, M.A. & Turbak, F. (2008). An aspect-oriented approach to the undergraduate programming language curriculum. *ACM SIGPLAN Notices,* 43(11). 124-129.

Silverstein, M. (1985). Language and the Culture of Gender: At the Intersection of Structure, Usage, and Ideology. In E. Mertz & R.J. Parmentier (Eds.), *Semiotic Meditation: Sociocultural and Psychological Perspectives* (pp. 219-259). Orlando, FL: Academic Press.

Smart, J. C., Feldman, K. A., & Ethington, C. A. (2000). *Academic disciplines: Holland's theory and the study of college students and faculty*. Nashville, TN: Vanderbilt University Press.

Spencer, S.J., et al. (1999). Stereotype threat and women's math performance. *Journal of Experimental Social Psychology,* 35. 4-28.

Stankiewicz, E. (1964). Problems in emotive language. In T.A. Sebeok, A.S. Hayes, & M.G. Bateson (Eds.), *Approaches to semiotics* (pp. 239-264). The Hague, The Netherlands: Mouton.

Stanley, J.M. *The Nerd Hour is at Hand: Portrayals of Geeks and Nerds in Portrayals of Geeks and Nerds in Young Adult Literature and Popular Media.* Master's Thesis. Retrieved from Digital Commons @ Longwood University.

Steele, C. M., & Aronson, J. (1995) Stereotype Threat and the Intellectual Test Performance of African Americans. *Journal of Personality and Social Psychology,* 69(5), 797-811.

Steffe, L. P., & Gale, J. (1995). *Constructivism in Education.* Hillsdale, NJ: Lawrence Erlbaum.

Stout, J.G. & Blaney, J.M. (2018). "But it doesn't come naturally": how effort expenditure shapes the benefit of growth mindset on women's sense of intellectual belonging in computing. *Computer Science Education,* 27(3-4). 215-228.

Svedin, M. & Bälter, O. (2016). Gender neutrality improved completion rate for all. *Computer Science Education,* 26(2-3). 192-207.

Talmy, S. (2008). The cultural productions of ESL student at Tradewinds High: Contingency, multidirectionality, and identity in L2 socialization. *Applied Linguistics, 29*, 619–644.

Tam, M.Y.S. & Bassett, G.W., Jr. (2006). The gender gap in information Technology. In J.M. Bystydzienski & S.R. Bird (Eds.), *Removing barriers: Women in academic science, technology, engineering, and mathematics.* Bloomington, IN: Indiana University Press. 199-133.

Tedre, M. et al. (2018). Changing aims of computing education: a historical survey. *Computer Science Education,* 27(4). DOI: 10.1080/08993408.2018.1486624.

Tobin, J. (2011). Save the Geeks. *Journal of Adolescent & Adult Literacy,* 44(6). 504-508.

Tonso, K. (2006). Teams that work: Campus culture, engineer identity, and social interactions. *Journal of Engineering Education, 95*(1), 25–37.

Toohey, K. (1998). "Breaking them up, taking them away": Constructing ESL students in grade one. TESOL Quarterly, 32, 61–84.

Toohey, K.: 2000, Learning English at School: Identity, Social Relations and Classroom Practice, Multilingual Matters, Clevedon, UK.

Tracy, K. (1997). Colloquium: Dilemmas of academic discourse. Norwood, NJ: Ablex.

UC Davis. (n. d.). Retrieved March 02, 2016, from https://www.ucdavis.edu/

Ulriksen, L. et al., (2015) The First-Year Experience: Students' Encounter with Science and Engineering Programmes. In E. Henriksen, J. Dillon, & J. Ryder (Eds.), *Understanding Student Participation and Choice in Science and Technology Education.* Dordrecht: Springer. 241-257.

Umapathy, K. & Ritzhaupt, A.D. (2017). A Meta-Analysis of Pair-Programming in Computer Programming Courses: Implications for Educational Practice. *ACM Transactions on Computing Education (TOCE)*. 17(4). Article 16.

Unfried, A., et al. (2014). Gender and Student Attitudes toward Science, Technology, Engineering, and Mathematics. *The Friday Institute for Educational Innovation at North Carolina State University*.

Urciuoli, B. (1996). *Exposing Prejudice: Puerto Rican Experiences of Language, Race, and Class.* Boulder, CO: Westview Press.

van Dijk, T.A. (1997). "The Study of Discourse." *Discourse as Structure and Process.* Teun A. van Dijk, Ed. London: SAGE.

Varma, R. (2010). Why so few women enroll in computing? Gender and ethnic differences in students' perception. *Computer Science Education, 20*(4), 301–316.

Veilleux, N., et al. (2012). The role of belonging in engagement, retention, and persistence. In *SIGCSE '12 proceedings of the 43rd ACM technical symposium on computer science education* (pp. 707–707). New York, NY: ACM.

Verenikina, I. (2004). From theory to practice: What does the metaphor of scaffolding mean to educators today? *Outlines: critical social studies, 6*(2), 5-16.

Vickers, C. (2007). Second language socialization through team interaction among electrical and computing engineering students. Modern Language Journal, 91, 621–640.

Vygotsky, L. S. (1978). *Mind in Society: The development of higher psychological processes* (M. Cole, V. John-Steiner, S. Scribner, & E. Souberman, Eds.) [Kindle DX version]. Retrieved from Amazon.com

Waite, W. M., Jackson, M. H., Diwan, A., & Leonardi, P. M. (2004). Student culture vs group work in computer science. *ACM SIGCSE Bulletin, 36*(1), 12–16.

Walton, G. M., & Cohen, G. L. (2007). A question of belonging: Race, social fit, and achievement. *Journal of Personality and Social Psychology,* 92(1), 82-96.

Webb, N.M. (2009). The teacher's role in promoting collaborative dialogue in the classroom. *British Journal of Educational Psychology,* 79. 1-28.

Weinberger, C.J. (2006). Earnings of women with Computer Science or engineering college majors. In E.M. Trauth (Ed.), *Encyclopedia of gender and information technology.* Hershey, PA: Idea Publishing. 224-227.

Wenger, E. (1998). *Communities of Practice: Learning, meaning, and identity.* Cambridge, MA: Cambridge University Press. [Kindle DX version]. Retrieved from Amazon.com

Werner, L.L., et al. (2004). Pair-Programming Helps Female Computer Science Students. *Journal of Educational Resources in Computing,* vol. 4.

Whiting, B.B., Whiting, J.W., & Longabaugh, R. (1975). Children of six cultures: A psychocultural analysis. Cambridge, MA: Harvard University Press.

Willett, J. (1995). Becoming first graders in an L2: An ethnographic study of L2 socialization. *TESOL Quarterly, 2*9(3), 473–503.

Williams, L. (2000). *The Collaborative Software Process.* PhD Dissertation.

Williams, L., et al. (2000). Strengthening the Case for Pair Programming. *IEEE Software,* 17(4). 19-25.

Williams, L.A. & Kessler, R.R. (2000). The Effects of 'Pair-Pressure' and 'Pair-Learning' on Software Engineering Education. *Proceedings of the 13th Conference on Software Engineering Education and Training.* 59-65.

Williams, L. & Kessler, R.R. (2002). *Pair Programming Illuminated.* Addison-Wesley Longman Publishing Co, Inc.

Williams, L. et al. (2003). Building Pair Programming Knowledge through a Family of Experiments." *Proceedings of the 2003 International Symposium on Empirical Software Engineering.* 143-152.

Williams, L., et al. (2006). Examining the Compatibility of Student Pair Programmers. *Proceedings of the AGILE 2006 Conference.*

Wilson, D. (2012). Metarepresentation in linguistic communication. In *Meaning and Relevance.* Cambridge, MA: Cambridge University Press. 230-258.

Winslow, L.E. (1996). Programming pedagogy—a psychological overview. *ACM SIGCSE Bulletin,* 28(3). 17-22.

Wong, B. (2016). 'I'm good, but not that good': Digitally-skilled young people's identity in computing. *Computer Science Education, 26*(4), 299–317.

Woolard, K. (1985). Language variation and cultural hegemony: Toward an integration of sociolinguistic and social theory. *American Ethnologist, 12,* 738-748.

Yim, Y.K.: 2005, Second Language Speakers' Participation in Computer-Mediated Discussions in Graduate Seminars, Unpublished doctoral dissertation, University of British Columbia, Vancouver, British Columbia, Canada.

Zander, C., Boustedt, J., McCartney, R., Moström, J. E., Sanders, K., & Thomas, L. (2009). Student transformations: Are they computer scientists yet? In *Proceedings of the fifth international workshop on computing education research workshop* (pp. 129–140). New York, NY: ACM.

Zappa-Hollman, S. (2007). Becoming socialized into diverse academic communities through oral presentations. Canadian Modern Language Review, 63, 455–485.