

PROGRAMMING AND CULTURE

A Dissertation

Presented to the Faculty of the Graduate School
of Cornell University

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

by

Ian Anders Arawjo

May 2023

© 2023 Ian Anders Arawjo

ALL RIGHTS RESERVED

PROGRAMMING AND CULTURE

Ian Anders Arawjo, Ph.D.

Cornell University 2023

I situate computer programming as a cultural practice. I develop this perspective in two ways: exploring how programming practices can support intercultural learning, and examining how programming tools themselves embed cultural assumptions and values. For the former, I study how relationships across difference are formed over computing activities in K-12 classrooms in Kenya and the U.S. Asking how programming concepts may serve people's intercultural development, I develop a new type of activity, "cultural algorithms," which uses algorithmic concepts to teach about the social construction of societies. Turning to the material means through which we 'write' code, I then trace the earliest history of programming and reveal epistemological tendencies and biases in the field. From the resulting insights, I develop a new AI-powered paradigm, notational programming, as one critical design that seeks to disrupt dominant norms around typing code. Throughout, I aim to muddle the boundaries between 'programming' and 'culture,' exploring programming both as a tool for making change (changing the programming in culture), and as a tool to be changed (changing the culture in programming). Ultimately, I argue that intercultural approaches to computing are focused on ontological change; that is, changing the boundaries and categories that people deploy to divide themselves from others and diminish the complexity of the world.

BIOGRAPHICAL SKETCH

Ian Arawjo grew up in Bethlehem, PA in the shadow of the Bethlehem Steel. He attended public schools and, as a teenager, got into trouble hacking into school laptops to gain admin privileges and install Halo: Combat Evolved. He later went to college in Montreal, Quebec at Concordia University for a double major in computation art and computer science. Many of his ancestors arrived to the U.S. at the turn of the 20th century from Poland, Ukraine, Hungary and Brazil, and worked manufacturing, clerical, or assembly-line jobs or held local furniture and tobacco shops in the PA area. His other lineage is to Pennsylvania Dutch settlers, specifically Mennonites, a Luddite ethnic group who persist today in rural areas of PA. He is the first person in his extended family to pursue and attain a doctoral degree.

To my parents, who never told me what to do.

ACKNOWLEDGEMENTS

It takes a village to raise a PhD student. The work herein could not have been possible without the generous support of many people along my journey at Cornell. In particular, I am indebted to my advisor, Dr. Tapan Parikh, for enabling my continued stay at Cornell, for supporting my (sometimes contentious) research pursuits, and for providing professional and personal guidance especially in the latter years of my PhD. I am also indebted to my committee, who supported, but never sought to interfere in, my research pursuits. In particular, I would like to thank Dr. Steven J. Jackson for the detailed direction he provided during the data analysis of the Kenya study, as well as for serving as a reader for the historical work you see in Chapter 5.

There are others who I would like to thank. My ongoing collaborations with Ariam Mogos, now a lecturer at Stanford University, have been a continual source of joy and learning –I confess that I probably learned more from Ariam than she ever learned from me. Without Ariam, the entire Part I of this thesis would not exist, as it would have never occurred to me to ask the research questions that underlie it. From Dr. Kentaro Toyama at the University of Michigan, I learned how to persist in the face of failure, as well as how to improve my writing process, both within academia and without. During my earlier PhD work on building an educational game to teach programming, my collaborations with Dr. Andrew C. Myers in the CS department were also a source of joy. Our conversations over the years have given me hope that researchers in programming language theory and researchers in HCI can, in fact, work together and learn from one another. There were also people that had a big impact on me, but whom are no longer in academia, that I wish to recognize. My previous advisor, Dr. Erik Andersen, took me on as a student when he was under consid-

erable pressure, and is a remarkable game designer. And some of the work you see here, particularly the Kenyan project, would not have been possible without the generous financial support of Dr. Arpita Ghosh, whom I collaborated with for 2+ years on economics research (alongside Dr. David Easley). Through that experience, I gained direct insight into the (often messy) world of proofs and mathematical models, an experience rare to researchers in science & technology studies that has become important to my interactions with researchers in programming language theory. I learned far more from Arpita than analyzing Nash equilibria in psychological games, however, such as the need to exercise the body, not just the mind, to improve one's mental health; or the importance of maintaining a clean work/life balance. I miss her presence greatly.

Finally, I would like to thank the many friends I met along the way: Matt, 'Gapo,' Sharifa, Ishtiaque, Swati, Neta, Hannah, Nathan, Alex, Daniel, Samir, JiYong, Vincent, Molly, Jingjin, Anthony, Mahima, Michael, and Basu. And, of course, I thank my parents and the wider Arawjo and Diehl families, whose support over the ups and downs of these many years has been immeasurable.

TABLE OF CONTENTS

Biographical Sketch	iii
Dedication	iv
Acknowledgements	v
Table of Contents	vii
List of Tables	x
List of Figures	xi
1 Introduction	1
2 Theoretical Background and Related Work	9
2.1 Defining Culture, Intercultural, and Intergroup	9
2.1.1 Culture	9
2.1.2 Intercultural	12
2.1.3 Intergroup	15
2.2 Motivating Educational Frameworks	19
2.2.1 Cultural-historical activity theory	19
2.2.2 Constructionism and cultural constructionism	21
2.2.3 Bennett’s Model of Intercultural Development	26
2.2.4 Critical Peace Education	28
2.3 Why “Intercultural” Computing?	29
2.4 Conclusion	31
I Programming and/in culture	32
3 Intro Programming Education as a Site of Intercultural Learning	33
3.1 Study Contexts, Programs and Methodologies	34
3.1.1 Nairobi Play Project	34
3.1.2 U.S. middle school study	41
3.1.3 Positionality and Collaborators	45
3.2 Findings from the Kenya Study	46
3.2.1 Computing Education in Support of Intercultural Learning	47
3.2.2 Overcoming Obstacles to Intercultural Learning	52
3.2.3 Complications of Intercultural Learning	55
3.3 Findings from the U.S. Study	57
3.3.1 Intergroup bonding over othering and stereotyping	58
3.3.2 Intergroup bonding with unequal control	60
3.3.3 Teacher discomfort around breaking up order	62
3.3.4 “Interdependent” programming	66
3.4 Overall Thoughts on Fostering Bonding Across Difference in Intro Computing Programs	71
3.5 Conclusion	74

4	Destabilizing Programming in Culture: “Cultural algorithms” & Inter-cultural Development	78
4.1	Racecraft and HCI	81
4.1.1	What is racecraft?	82
4.1.2	Why do we need another race framework?	86
4.1.3	The performativity of “race is socially constructed”	89
4.1.4	Past work in HCI, critiqued through the lens of racecraft	96
4.2	Birdcraft: a cultural algorithm activity for racial literacy	104
4.2.1	The Design of Birdcraft	104
4.2.2	Deploying Birdcraft with participants	109
4.2.3	Reflections	118
4.3	Conclusion	120
 II Culture in programming		 121
5	The Origins of Programming as Cultural Activity	122
5.1	The Paradox of Change in HCI	124
5.2	Human-Machine Interaction Before HCI	128
5.3	The Culture ‘in’ Early Programming Notations: Three Visions	131
5.3.1	Konrad Zuse’s Vision	132
5.3.2	The ENIAC Vision	135
5.3.3	The Typewritten Vision and the Serialization of Programming Notation	139
5.4	Discussion	143
5.4.1	Framing the Early History of HCI as Situated Knowledge and “Making Do”	145
5.4.2	The Naturalization of the Textual/Visual Dichotomy	147
5.4.3	Embracing Heterogeneity in Programming Practice	148
5.5	Conclusion	151
6	Destabilizing Culture in Programming: The Case of Notational Programming	153
6.1	Related Work	156
6.1.1	Programming Systems and HCI	156
6.1.2	Pen-based interfaces for programming	158
6.2	What is Notational Programming?	159
6.2.1	Definition and principles	161
6.3	Case study: The Qaw Quantum Circuit Notation	165
6.3.1	Notation Design Process	168
6.3.2	Examples: Superdense Coding & Grover’s Algorithm	169
6.3.3	Implementation	171
6.4	Usability Study	172
6.5	Findings	175

6.5.1	Conceptual (mis)understandings	176
6.5.2	Error handling and debugging	181
6.5.3	Interactions with hardware and software	185
6.6	Comparison with a Typewritten API	187
6.6.1	Participants	188
6.6.2	Task by Task Performance	188
6.6.3	Qualitative observations	192
6.6.4	Limitations of evaluation	194
6.7	Discussion	195
6.7.1	Future directions and reflections on process	195
6.7.2	Rationale behind turns of phrase	197
6.7.3	Handwriting interfaces vs. GUIs	199
6.8	Conclusion	201

III	Conclusion	203
7	The Future of Programming and HCI	204
	Bibliography	212

LIST OF TABLES

6.1	Some common elements of notation that practitioners use to write quantum circuits. For a full description of notation included in Qaw, see Appendix A. For an intro to quantum computing, see the Qiskit textbook [210].	166
6.2	Post-survey Likert results for our Notate user study. Shapiro-Wilk tests indicate non-normal distributions at $p < 0.05$ for all questions except Q5; hence, I report medians. Variances are high, consistent with programming studies [89].	176
6.3	Task times (\tilde{x} =median, s =st. dev) and comparison statistics (Mann-Whitney U, Levene with median) across conditions. ($*$ = $p < 0.1$ indicating a potential trend, $**$ = $p < 0.05$ indicating significant, $***$ = $p < 0.01$ indicating highly significant)	187

LIST OF FIGURES

3.1	Activities depicted	48
3.2	Group game designs embody experiences shared across difference: (a) a boy avoids mosquitoes to reach red malaria pills; (b) a woman flees across a border as “terrorists” block her path; (c) a student vies to finish their education by answering questions on exams and dodging distracting mobile phones.	51
3.3	During a pair programming activity, a hispanic girl and a white boy create a game about helping an old woman cross the street. The scene ends with the old woman thanking the man and the man replying “vale Boomer” (“OK Boomer” in Spanish). The pair were observed giggling together over the project, but the U.S. teacher, an older male near the boomer generation, reacted negatively to the epithet.	58
3.4	Two partners –a black girl, Jordan and a white boy, Evan –create games of each other’s heroes and the obstacles they faced. The depicted games are the final versions.	67
4.1	The worksheet for Birdcraft, depicting Hoyt’s 5 steps of racialization as pseudocode.	105
5.1	Excerpted notations of German logicians Hilbert/Ackermann and Frege, whose works Zuse studied. Below, a handwritten excerpt from Zuse’s 1945 Plankalkül notebook, showcasing the <i>Zeilenverschiebung</i> , or ‘line shift’ notation [448]. <i>From original texts</i> [198, 147, 449].	133
5.2	(a) Section of ENIAC accumulator block diagram abstracting an electronic circuit, Arthur Burks, Aug. 1947; (b) Section of flow diagram hand-drawn by Adele Goldstine, Dec. 1947. <i>Rewritten from originals</i> [70, 162].	136
5.3	Examples of translations from mathematical notation to what FORTRAN designers anticipated could be typed on an IBM key-punch (1954). Notice the handwritten \times symbols and lowercase letters: by 1956, these became asterisks and uppercase letters [34, 35]. <i>Rewritten from original</i> [36].	140
5.4	Jottings from John Backus, written on a note while at the Arlington Hotel in Binghamton, NY, depicting translations between mathematics (bottom row) and what can be typed on an IBM keypunch (top two rows), likely in preparation for a talk. Note the inconsistency in notation for definitions sumA and sumB: dots and exponentials could not be typed. <i>Photo of original</i> [31]. .	142

6.1	Circuit equalities from the advent of computing to quantum computing. Left: A circuit equality written by von Neumann in <i>First Draft of a Report on the EDVAC</i> , 1945 [415]. Right: a circuit equality from a quantum computing text.	156
6.2	The main interface to the Notate system, embedded in a Jupyter notebook: (1) a canvas torn open inside a line of code in a cell; (2) fullscreen mode, accessed by touching or clicking on the canvas, with (3) a rudimentary toolbar.	159
6.3	A user copies an image of a quantum circuit from search results (1) and pastes it directly into a function call (2). The user runs the cell and views output (3), verifying that the interpretation is correct.	160
6.4	In a code cell, a user draws a diagram to calculate the value of angle a between two 2d vectors, b and c , defined as tuples in Python code. The Interpreter <code>geo</code> takes a Canvas and (implicitly) a reference to the local scope. Interpreting the diagram, it associates label a with an angle, realizes that a is not set, and declares it as a new variable in the host scope.	160
6.5	In a code cell, a user draws a diagram without prespecifying values for sides a , b and c . The diagram returns a queryable object. By setting some parameters (here, the <code>ask</code> method), the object returns the value of the remaining unspecified length. Note that this example is meant to express the possibilities afforded by notational programming, not argue the quality of this particular example's API design.	163
6.6	A common circuit for superdense coding, handwritten in Qaw (top) with Qiskit code for comparison. Notable features: 1) circles mark classical control bits a and b 2) kets initialize qubit lines, 3) stoppered outputs represent 'measure' operations. Here a, b variables are implicitly referenced in the handwritten context as classical bits that control gates.	170
6.7	Top: a way to write Grover's algorithm, from the Qiskit textbook, depicting the slash notation used in many quantum computing resources [210]. Bottom: the above circuit "coded" in Notate with a version of the Qaw notation, where \mathfrak{D} is diffusion circuit and \mathfrak{U} is the oracle (to be defined). The \mathfrak{D} circuit is a solution to Task 3 in my user study.	171
6.8	The process of interpreting a handwritten quantum circuit in my system. (<i>Drawing is P14's solution to Task 1.</i>)	172
6.9	P5's solution to Task 6, generating a pattern similar to the the body of the quantum fourier transform. Circuits \mathcal{A} and \mathcal{C} are defined recursively using Qaw slash notation and implicit cross-context references.	180

6.10	Three common “syntax errors” made by participants: (1) missing an output wire to a gate; (2) slashing an output wire of a control gate; (3) trying to use a single dot to represent a multi-Controlled Z gate in Task 3. In future iterations, we can imagine supporting some of these styles.	182
6.11	During Task 5, P8 encounters an error plot thrown by the AI recognizer (below the code cell). Noticing that a Z gate was mistaken for a B, P8 responds by erasing and rewriting her Z, then guessing the open corner was also an issue (“ <i>Maybe it’s also this thing over here?</i> ”) and filling it in. She runs the cell; it throws another error plot. The researcher present remarks on the $n-2$ as a conceptual misunderstanding –this notation was only used in a tutorial to explain how a recursive definition unrolls. P8 erases it and runs the cell, leading to the expected output (“ <i>Alright! Looks good.</i> ”).	185

CHAPTER 1

INTRODUCTION

Computing is at a crossroads. Where before humanistic, critical approaches from science and technology studies (STS), anthropology, cultural studies and more were eschewed as outside the “core” of computer science (CS) and human-computer interaction (HCI), increasingly, the image of computing as a purely “technical” field is beginning to fall. Alongside this shift that we might call computing’s “loss of innocence” or “loss of naiveté” –embodied in the foci of ethics and fairness, structural discrimination and inequality, and the limitations of technological solutionism –is a growing attention towards history and cultural difference, particularly sociocultural theories that mobilize words like “ontology” and “epistemology” in what were heretofore technical conferences or journals.

The field of computing, therefore, no longer knows what it is —or, at least, has less of a coherency about it than ever before. We now approach new technology with hesitation, disillusioned by failed deployments or deployments that have worked so well that they have amplified inequality and division, while leaving the underlying architecture of society untouched. When marketers claim a new technology will solve pressing problems or meet dire social needs, we now expect the opposite —that more Facebook friends and engagement has no relationship to quality of social life (perhaps even an inverse one); that Twitter drives us further apart; that dreams of space travel are yet another diversion for capitalists to avoid redistributive change. For HCI researchers, it is increasingly commonplace, even expected, to be a cynic.

My dissertation concerns one corner of our newfound interest in (or rather,

return to) questions of culture and history in computing: the programming of machines and the practices and notations used to accomplish it. Programming has become so commonplace, so normalized, its practices so deeply entwined with standardized hardware (QWERTY shift-key keyboards, mice, etc.) and software (command line, IDEs, etc) that it has become exceedingly difficult to imagine it otherwise, to see it not as a set of inevitable practices that emerged from logic and reason, but culture- and history-laden practices that involved, and in some cases reproduced, existing norms, values, and beliefs of their progenitors.

In this thesis I argue that computer programming should be seen and engaged through a cultural lens. I consider programming activities as sites of inter-cultural encounter, conflict, and positive transformation. I show how programming practice itself arose from such encounters, and how the early history of programming inscribed and reinforced certain epistemological perspectives over others. Throughout, I aim to muddle the boundaries between “programming” and “culture,” exploring programming both as a tool for making change (changing the programming in culture), and as a tool to be changed (changing the culture in programming).

My situating of programming as a cultural practice —or as it is fashionable to call it today, “coding” —is not new. It meshes with some early thoughts on programming that curiously came not from the technologists or CS professors per say but from the learning sciences, where for several decades academics have engaged questions at the intersection of computer programming and culture, often derived from intractable inequities. The work of these scholars was not just a matter of being culturally responsive in specific activities, but working

to change what Turkle & Papert called the “computing culture” [407].

Turkle & Papert’s work on epistemological pluralism raised questions about how the cultural backgrounds and interests of novices informed their programming practice. Some of the emergent conflicts between teachers and students were not conceptual misunderstandings but rather cultural, about valuing certain ways of coding over others. In the process of translating the practice of poetry into a computer program, for instance, students encountered a deeper conflict between “general-purpose” practices which dictated how they “should” program, and how they preferred to program. These dominant ways of programming had been *codified*, encoded into the programming culture (and sometimes, the design of its notations and software) from people who had different goals (e.g., plotting missile trajectories), backgrounds, and identities. Thus, this early work expressed the following insight: that asking questions about the culture of programming inevitably leads us to question how culture has been programmed, by whom, and for what ends. As computer programs embed human rules and procedures, so do people act on and transmit rules and scripts; e.g., inheritance law, voting procedures, academic credentialing processes, and racial ideologies. Ultimately, when we are interested in the culture in programming, we are interested in the programming in culture.

This dissertation has two parts. The first half concerns how programming activities can support people’s intercultural learning. It begins with work on how to bring students together across difference in intro CS classrooms at the K-12 level, and leads into exploratory work on how algorithmic thinking can aid the development of intercultural competence, through the concept of “cultural algorithms.” The cultural algorithms section focuses on intercultural learning

about racial ideology, by introducing *racecraft* as a intercultural perspective on race/ism in the U.S. [142], and then exploring an example activity, Birdcraft, that uses pseudocode as a tool to help people understand it. The second half of this dissertation concerns the culture “in” programming (communities and tools) through historical research and critical design. It traces the earliest history of “writing code,” and afterwards explores a new paradigm of interaction, *notational programming*, which seeks to destabilize the dominant, typewritten paradigm. I summarize these chapters below.

Chapter 2 overviews prior work and relevant concepts throughout the thesis, including culture, intercultural, intergroup, cultural-historical activity theory, constructionism, and critical peace education.

Chapter 3 draws from empirical work to explore contemporary computer programming spaces and practices as sites of intercultural encounter and development. Instead of viewing K-12 CS classes as places for ingesting concepts, or even exploring individual identity through project-based learning, I ask how it might be a place for intercultural learning: bridging social divides, developing intercultural competence, and facilitating new intergroup friendships which outlast classes. Drawing from qualitative studies of intro programming classes for youth in Kenya and the U.S., I suggest computing education’s affordances for intercultural learning and intergroup bonding, as well as several complications which strain normative assessments of learning outcomes or present dilemmas to educators and researchers (e.g., tensions between equity goals and intergroup contact).

Chapter 4 asks how we might leverage the concept of an *algorithm* to teach about the construction of societies; specifically, to denormalize tacit cultural

practices and beliefs [183]. This contrasts with prior work that focused on more “explicit” connections of computing to culture through craftwork, language or fashion (e.g., ethnocomputing, culturally responsive computing). Drawing from a workshop conducted with K-12 CS educators, I introduce and explore the concept of a “cultural algorithm” as a mechanism to help participants understand social constructionism, specifically the development of intercultural competence around “race” and racism. I situate this perspective primarily in the work of Black and Indigenous scholars and intellectuals, foremost the Fields sisters’ *racecraft*, as well as Toni Morrison’s concept of the “racial house,” Paul Gilroy’s position “against race,” and Carlos Hoyt’s “non-racial worldview” [286, 152, 206, 142]. Far from being separate topics, I suggest that computing concepts can contribute to future anti-racist action by revealing the hidden gears of racecraft and foregrounding how racism produces race.

Chapter 5 shifts towards material concerns to trace how programming has always been inter-cultural —a practice that lies at the intersection of various cultural practices —by providing a historical analysis of the earliest history of electronic computer programming. I show how power dynamics and sociomaterial forces shaped the dominant computing culture that stabilized around the mid-1950s. In particular, I argue that the dominant programming practice which arose around FORTRAN re-inscribed a prior European division between writing and drawing, the “textual” and the “visual.” My historical work provides a social and, crucially, material foundation for conceptualizing sites of computer programming today as opportunities for either embracing, or diminishing, the cultural differences that people bring to the task. We see how the personal intercultural development (or lack of it) of key actors in computing history helped shape the field in critical ways, in what value they attached to certain represen-

tations over others. In particular, I suggest that the term “language” became attached to “programming” exactly when the typewriter was adopted as the primary input method. Because of this association between language and typing in programming practice, I suggest that whenever we center the term “programming language” (over, say, programming environments or systems), we are implicitly recentring typing as the dominant input method.

Chapter 6 expands upon points raised by my historical analysis in Chapter 5, in particular the claim that programming’s early history re-inscribed an artificial division between “writing” and “drawing” (and relatedly, the “textual” and the “visual”). In that chapter, I concluded that future tools should seek to challenge the writing/drawing dichotomy and centrality of typewritten input. To explore one design that embodies this alternate reality, I develop a new programming paradigm, *notational programming*, that integrates handwritten and typewritten notations within a traditional code editor. I build a prototype, Notate, which allows users to open drawing canvases within lines of code. I describe mechanisms of how handwritten and typewritten practices might work in concert, developing an interaction principle, “implicit cross-context references,” where handwritten symbols can refer to typewritten ones and vice-versa. As a case study, I explore quantum computer programming and extend quantum circuit notation with abstraction features, including variable-sized inputs and recursion. Using deep learning and classical sketch recognition techniques, I then implement an interpreter for a subset of this new notation. A usability study suggests that participants find the core interaction of “implicit cross-context references” intuitive. A further comparison with a typewritten API shows that, in spite of troubles with recognition rates, handwritten notation was, in the large, comparable in terms of task time to typing code. I suggest that one medium

is not universally better than another, and advance a “heterogenous” vision of programming as the intermingling of type- and hand-written practices.

Chapter 7 (Conclusion) ties it all together by concluding that intercultural approaches to computing are interested in ontological design [429, 433, 137], or rather, changing or challenging dominant ontologies that enforce rigid regimes of classification which divide and diminish. Threading my work is how computing, and its rich concepts and spaces, might be tools both for making change, and tools to be changed. To close, I suggest that resisting dominant social practices often requires actions and design choices that are initially perceived less favorably: whether upsetting established dogma, or denigrated as naïve. Rather than seeing such reactions as negative, however, I conclude that they may serve as important signals that one is on the right track.

This thesis takes a breadth-first approach to its investigations of programming as a cultural practice. It applies methods ranging from ethnographic and qualitative, to archival research, to usability studies with mixed methods.¹ It bears mentioning that this approach was not planned or chosen. My initial goal was for the content of Chapter 3 to form the bulk of the dissertation, but the advent of the COVID-19 pandemic disrupted my educational research in schools. In response, the thesis necessarily had to widen in scope, resulting in Chapter 4 on “cultural algorithms” activities and theorization, as well as the final chapter about the design and evaluation of a deep learning-powered system for “notational programming” (a practical offshoot of my historical work in Chapter 5).

¹The research is also interdisciplinary, connecting to literature in a wide range of fields, from information communication technology for development (ICTD), to the sociology of race/ism and ethnicity, to quantum programming. Although theoretical contributions are not the primary focus, critical HCI researchers may find some useful nuggets here; for instance, I take issue with widely read works in the field, such as claims by Ames about the One Laptop Per Child program embodying Papert’s constructionism, or Chun’s claim in “Race and/as Technology” that racism “stems from race” [12, 86]; see Chapters 2 and 4.

I believe the thesis is stronger because of this circuitous path, instead of in spite of it. Throughout, I hope readers find interesting takeaways that prove useful for those seeking to understand –and sometimes to challenge –the taken-for-granted practices of computer programming.

CHAPTER 2

THEORETICAL BACKGROUND AND RELATED WORK

My work draws from a wide range of theoretical frameworks and literatures, including cultural-historical activity theory, intercultural development theory, prejudice reduction, critical peace education, the sociology of ethnicity, and cultural constructionism. In this chapter, I provide an overview of key terms and frameworks that shall appear throughout the rest of this thesis. These are roughly divided into topics of culture, interpersonal relations, and pedagogy. The order is strategic: the “cultural” foci frames the latter two concerns, and our understanding of relationships across difference informs pedagogy. After exploring these three topics, I briefly justify why I chose the term “intercultural” over “intergroup” for my educational work.

2.1 Defining Culture, Intercultural, and Intergroup

2.1.1 Culture

As might be anticipated, “culture” has a long history as a term. Decades of scholars have grappled with the, in the words of Edward Hall, “very muddied” concept [183, p. 43]. Here I attempt a brief summary, leading into the emergence of the “intercultural” term.

Evolving from the original Latin term *cultura* meaning cultivation of soil, “culture” became an agricultural metaphor for tending the “soul” or mind around the early 19th century [411]. This concept of culture was linked to upper-

class interests such as the arts, music, fashion, literature, and theatre and meant “an institutional sphere... devoted specifically to the production, circulation, and use of meanings” [376]. Although embraced by early sociologists and cultural studies scholars, anthropologists rejected this “high-class” concept of culture and developed instead a “structuralist” or functional concept of culture as a bounded, static, closed property of a community which explains its inner workings and the sustainability of its way of life. The structuralist concept would receive its own critique around the 1970s-1980s, however, when some anthropologists argued for a more “dynamic” concept of culture; this resulted in furthering the divide between “culture as a system of symbols and meanings” and “culture as practice” (p. 44-46). Critiquing the former concept and concerned with the colonial underbelly of early anthropology, some anthropologists proposed doing away with “culture” entirely, preferring alternative terms such as “practice” and “discourse” [3]. For instance, the term “culture” is notably absent in Donna Haraway’s celebrated article “Situated Knowledges,” even though what she often seems to be speaking about is cultural difference [192].

Despite these clarion calls, “culture” is not dead. Coming from a linguistic perspective, Brightman suggested that many of culture’s detractors in anthropology constructed a “straw culture” to attack by cherrypicking certain historical uses while avoiding others; for Brightman, culture had always included both “practice” and “system” concepts, and close readings of classic texts would reveal this [64]. Sewell agreed and called instead for a merging of definitions: “[if] we cannot do without a concept of culture, I think we should try to shape it into one we can work with. We need to modify, rearticulate, and revivify the concept, retaining and reshaping what is useful and discarding what is not.” Culture is system and practice for Sewell, “worlds of meaning as normally... contradictory,

loosely integrated, contested, mutable, and highly permeable” [376, p. 38, 53].

Indeed this definition is closer to how “culture” is usually mobilized in the learning sciences, anthropology, and HCI today [176, 94, 215, 77] —as “generative” or hybrid rather than static, encompassing the arts and practice, thought and action. For instance, Irani & Dourish resist “taxonomic” definitions of culture, e.g. those which see culture as a system of classification and geographic separation (such as Hofstede [201]), and instead ask how we might account for “global traffic in cultural concepts.” They define culture as “a lens through which people collectively encounter the world, a system of interpretive signification which renders the world intersubjectively meaningful... [A]n individual may participate in many cultures —cultures of ethnicity, nationhood, profession, class, gender, kinship, and history —each of which, with their logics and narratives, frame the experience of everyday life” [215, p. 2-3]. Where before culture referred to, say, artwork or religion, now “culture” also comprises both mathematical notation and the “feel” of Silicon Valley organizations, thanks in part to science and technology studies’ (STS) reframing of culture as something that occurs also in scientific settings [331]. Irani & Dourish’s focus on how the “intersubjective” constructs difference indeed sounds like social identity theory (a point I will elaborate on shortly); an Oxford encyclopedia article around prejudice and discrimination defines a similarly inclusive definition of culture as: “The way of life of a group of people, including symbols, values, behaviors, artifacts, and other shared aspects, that continually evolves as people share messages and is often the result of a struggle between groups who share different perspectives, interests, and power relationships” [40].

These definitions, bundling both “system” and “practice” into one concept,

do avoid the trap of seeing cultures as well-bounded and mutually exclusive, but may cause us to wonder what, in fact, is not cultural! If we hope the learning sciences will fish us out, we are out of luck; cultural-historical activity theory (CHAT) has perhaps the most generously expanded definitions of culture.¹ Despite this ambiguity, “generative” interpretations of “culture” are the ones I shall adopt for this thesis. The postcolonial definition offered by Irani & Dourish, coupled with an emphasis on artifacts and theory of cultural learning, captures the cultural factors that computing education involves—heavy use of artifacts, technologies, epistemologies usually developed in a Western context; cooperation and problem-solving across differences; shared design and negotiation. Rather than the taxonomic approach to culture, individuals are viewed as representing a potential plurality of cultural expression and understanding. This opens the door to a generous “intercultural” framing [215] or “cultural flexibility” [77] that also includes material and environmental factors. My generative usage of “culture” is meant to be an orientation or sensibility towards deferring, as much as possible, to how cultural difference emerges in classes, rather than prematurely imposing assumptions about particular groups or categories, and therefore aligns with an ethnomethodological or social constructivist approach.

2.1.2 Intercultural

Given that we already have a term for “culture” that is rather broad, why introduce yet another? The difference may be on emphasis: historically, the term “intercultural” centered interactions across (usually substantial) cultural differences, rather than analyses of general cultural practices and systems within a

¹See, e.g., Gutierrez et al. [176]. I will expand upon CHAT in Section 2.2.1.

particular society, community or discipline. The term was popularized by the field of intercultural communication, founded by Edward Hall in work at the Foreign Service Institute at the U.S. Department of State in the 40s and 50s [250]. Since then, there have been many definitions and subfields using the term “intercultural,” succeeded by multiple nouns: “understanding,” “dialogue,” “learning,” “competence,” etc. (the issue is compounded when we consider the ambiguously related field of “cross-cultural” factors). Here I briefly summarize the history and general usage of “intercultural.”

Edward Hall’s book *The Silent Language* [183] introduces the general problem of communication across what at that time were, at least much more so than today, largely geographically-bounded societies with their own distinct differences —through language, ways of dress, religions, etc. —applying a similar definition of culture to that of anthropology of the time. *The Silent Language* was preoccupied with tacit behavior and value differences between cultural groups along the lines of “time and space” (p. 24). Intercultural learning for Hall was about teaching how concepts of time and space operate differently to different communities of people, and how these tacit differences are the source of much conflict. Culture for Hall was “a form of communication,” (p. 51) verbal and non-verbal, explicit and tacit (in Hall’s terms, “formal, informal, and technical”); culture “speaks” (p. 55). Hall describes conflicts between two interacting societies that might otherwise be avoided, should one side be trained in cultural differences and self-reflection on their own background. Examples were often along the lines of white employers looking to speak to and manage “native” employees, or U.S. diplomats in negotiation with Asian societies. Hall also seemed to draw an implicit line between inter-“racial” conflicts present in U.S. settings and his “intercultural” conflicts or misunderstandings abroad. For instance,

Hall introduces the book with a story about majority-minority conflict during a stint as “a member of a mayor’s committee on human relations in a large [U.S.] city.” In interviews with administrators, he describes how their response to a non-discrimination policy could be predicted by whether they kept him “at a distance” during the interview or not, operationalized through space (chair-to-chair distance or obstacles between them) or time (appointments lost, or little time set aside to discuss). Although Hall’s “formal, informal, technical” taxonomy of culture does not seem to have taken root, the idea that different societies invent tacit, not just explicit, means of communication (that insiders struggle to reflect on), and that these tacit differences are the source of much “inter”-cultural conflict, persists today (often spoken of as conflicts between different ontologies and/or epistemologies, see Verran [412]).²

By focusing on situated interactions and communication, Hall took an “emic” (insider’s point of view) perspective rather than the “etic” (high-level comparative analysis) perspective common to anthropology of the time [250]. Hall’s approach necessitated a shift in how anthropologists dealt with “culture” from their “macro” analysis to a micro-sociology of interactions, grounded in the need to give foreign service diplomats and corporate actors practical advice on everyday communication.³ In this way, intercultural communication seems to have paved the way to the expansion of the “culture” concept similar to the generative and CHAT understandings previously mentioned. For instance, in a more recent article on intercultural competence, Kim defines cul-

²This is in sharp contrast to Tajfel’s social identity theory, which was not a theory of communication but rather provided a partial psychological explanation for out-group prejudice, founded on economic experiments. I will cover Tajfel’s work shortly.

³Measures that have been developed for intercultural factors reveals the fields’ continued preoccupation with adult, business settings [274]. For instance, the intercultural development inventory (IDI) requires certification, requiring thousands of dollars to attend a training workshop and be certified.

ture as “not only to the shared life patterns within a society or a nation, but also to those patterns associated with distinct ethnic (including national, racial, religious, and language-based) groups within a society” [238]. According to this definition, “any interpersonal encounter is considered ‘intercultural’ whenever the interactants differ, or perceive themselves to be different from each other, in cultural or subcultural backgrounds... two core terms, intercultural conflict and intercultural affiliation, are likewise employed to represent related terms such as ‘interethnic,’ ‘interracial,’ and ‘intergroup’ conflict and affiliation.” Of key interest here is the phrase “perceive themselves to be different,” which harkens to the intergroup literature: whereas Hall studied communication between different, largely geographically or ancestrally separate peoples, today the term ‘intercultural’ includes people who may not be very different but merely perceive themselves to be.

2.1.3 Intergroup

The term “intergroup” was popularized by social psychologist Henri Tajfel’s social identity theory, introduced in a paper in the journal *Social Science Information* and elaborated from a series of experiments presented in *Scientific American* [392, 393]. Tajfel was concerned with how individual, psychological processes interacted with the phenomenon of out-group attitudes and behavior found across many societies “be they racial as in the U.S., religious as in Northern Ireland or linguistic-national as in Belgium” [392, p. 96]. To study the effects of categorization and in-group identity on intergroup attitudes, Tajfel’s “minimal group” experiments showed how the mere act of (explicit) categorization produced in-group bias. The experiments involved a homogenous set of par-

ticipants (e.g., all males from a dormitory in Bristol, UK), and two phases. In the first, participants were categorized into two groups according to “guessing numbers of dots... or expressing preference for the paintings of one of two fairly abstract painters” [393]; in the second, they allocated money to randomized, anonymous peers whose only identification was group affiliation. A clear significant and reproducible in-group favoritism was observed. Results were significant in that, unlike prior work, great efforts were made to isolate confounding variables; “subjects were never together as a group; they neither interacted nor did they know who was in their own group and who in the other; there were no explicit social pressures on them to act in favour of their own group; and in no way was their own individual interest engaged in awarding more money to a member of their own group” [393]. Building on this seminal study, Tajfel hypothesized that individuals have a psychological need to categorize peoples (including themselves) into group affiliations in order to maintain social cohesion and that this need in part contributes to in-group favoritism and out-group attitudes and behavior, apart from any historical or “objective” meaning for conflict. Interestingly, the original 1970 paper attaches this need to “social norms” present in “most modern societies,” showing Tajfel uneasy to cast “groupness” as a cognitive universal (e.g., Tajfel uses the phrase “supposedly universal human drives” [392]). Grounded in categorization and economic, quantitative experiments, social identity theory is thus “not a communication theory,” but rather represents a theory of social behavior and cognition [150].

Related to, and in many cases growing out of, social identity theory is a broad literature studying the cognitive basis of bias and prejudice. The social psychologist Mahzarin Banaji, for instance, advanced the notion of “implicit bias,” prejudices one holds subconsciously towards individuals one perceives to

be members of a social group [170]. Prejudice reduction experiments, whether minimal group, implicit bias, or other types of experiments, form the bedrock of the prejudice reduction literature and provide cognitive basis for prejudice and the role of categorization in discrimination [68]. Yet as a review of the prejudice literature remarks, it is unclear how directly controlled studies correspond to conflicts in the wild [313]. As sociologists and anthropologists have argued [45, 68], the difference between artificially constructed groups and socially constructed, pervasive categories is that identifications are static in experiments, whereas they are dynamic and co-constructed in real life. Conflicts in the wild often emerge around the boundary of the category's definition, a difficulty not usually captured in experiments where groups are created on-the-spot. Moreover, while some believe implicit bias is at the heart of "microaggressions" towards individuals perceived to be members of non-dominant social groups, some studies conflict with this intuition, showing no correlation between implicit bias and explicit discrimination [313].

Many interventions aimed at bridging differences and reducing prejudice are based on the intergroup literature. One of the most prominent practical frameworks has been intergroup dialogue (ID) programs. Developed at the University of Michigan in the late 1980s in the context of a waning civil rights movement [175], ID is "a facilitated group experience that may occur once or may be sustained over time and is designed to give individuals and groups a safe and structured opportunity to explore attitudes about polarizing societal issues" [118]. The approach has spread across U.S. universities; for instance, Cornell has its own version. Informed by extensive empirical studies in prejudice reduction and multicultural education, ID programs at large adopt a "dual category" approach where both subordinate and superordinate identities are

recognized and emphasized (e.g., one's racial or ethnic identification and their shared belonging to a school) [313]. For instance, Gurin et. al's study bifurcated participants 50/50 between "white" and "people of color" and based statistical analyses, as well as intragroup dialogues, on this separation [175]. Intragroup dialogues are seen as important for preparing students for intergroup dialogue and mitigating, e.g., anger, silencing, and other emotional behavior that shuts down dialogue prematurely. A multi-institution, longitudinal, mixed methods study by Gurin et al. demonstrated the method's success in reducing prejudice and increasing empathy between university students. As I shall argue in Chapter 4 in reference to the Fields' concept of *racecraft*, these intergroup approaches have the potential risk of reifying static and ontological notions of race that have been critiqued by other scholars [206, 152, 142]; in particular, there appears nothing strange about the act of "racial" sorting into *a priori* identity groups, whether to the participants or the researchers themselves.⁴

The challenge of ID programs is that facilitators must be highly trained or else reverse outcomes may occur. This is especially the case since intergroup dialogue, by contrast to, for instance, sports for development programs, has no external task that distracts from tension. Yet facilitators often either do not have the time for adequate training or their methods vary widely [118, 171]. This is a similar issue to diversity training for large organizations, where the superficial nature of lessons and/or facilitator skill has been shown to lead to nonexistent, often even negative, outcomes across two large literature reviews [55, 120]. These issues reflect the dangers of learning about power differences and social groups that does not involve deeper empathy and understanding

⁴This is analogous to the similar reifications present in HCI; e.g. in Kleinberg et al. [240], whose mathematical analysis of affirmative action assumes the goal is the "equality;; of racial groups—a goal that appears absurd when we realize that race is constructed through asymmetric rules on descent. See Chapter 4.

between individuals but rather serves to stereotype certain groups and advance a legal agenda by creating the appearance of a commitment to diversity and inclusion [120, 409]. It is important, as we will see, that intergroup conflict is seen as mutually constituted (as Tajfel originally posits [393]).

Connecting back to the intercultural literature, there appears to be a large overlap between studies mobilizing the terms “intercultural” and “intergroup” today [150], as deficiencies in one theory seem to be resolved by reference to the other (for intercultural field learning from intergroup, see [76]; for the reverse pollination, see [382, 175]). Thus, we may hypothesize that these fields study fundamentally similar phenomena, but were biased by their original context, methods, and analytic focus [150], and are therefore developing asymptotically towards one another similar to how the fields of inter-racial/ethnic/national conflict overlap [68].

2.2 Motivating Educational Frameworks

Having reviewed key terms of “culture,” “intercultural” and “intergroup,” I now cover four main literatures to ground the educational aspect of intercultural computing: cultural-historical activity theory, constructionism, peace education and prejudice reduction, and models of intercultural development.

2.2.1 Cultural-historical activity theory

A body of early work in CSCW and CSCL applied cultural-historical activity theory [228] (CHAT) to the design of educational innovations involving technol-

ogy. Pioneered by Russian psychologists Vygostky and Leont'ev, and extended by scholars such as Engeström [132], Cole [94], and Gutiérrez [177], CHAT emphasizes learning as a process of social and cultural activity, rather than individual construction, in which “diversity is [viewed as] a resource and heterogeneity is a design principle” [177, p. 216]. These approaches gained particular currency in literacy education to address or account for apparent disparities without resorting to casting certain cultures as “deficit” [177, 94].

A prime application of the CHAT approach can be found in the Fifth Dimension after-school program [94], which spanned twenty years of research and involved multiple institutions, academics, and community stakeholders. A central goal of the Fifth Dimension was diversity in all respects—in the available activities, in the participants (at all levels, from student to institutional), and in the adaptability of the innovation to local context. Michael Cole, the leader of the project, noted that there are two prevailing views on diversity —“make it go away” (through assimilation) or “make use of it” as a “resource” for reciprocal exchange [93]. A decade after this statement, a review by UNESCO of sustainable development programs found that cultural differences remain all too frequently “interpreted as constraints to progress towards sustainable development... [i]ntercultural dialogue is rarely seen as an opportunity to explore new creative ways to live or construct a sustainable future amongst diverse groups” [400].

CHAT has arguably three tenets: first, internalization/externalization, which accounts for how cultural practices are internalized and become tacit, while the process of their externalization can help create new artifacts/tools or methods (Engestrom 1999). Second, mediating artifacts/tools take center stage

in CHAT. Learning always arises from sociocultural origins, with particular emphasis on relationships between people and social transfer of knowledge (e.g., peers or teacher-peer) through situated activity. Third (and arguably the most important principle for this thesis) is CHAT's reframing of normative conflict and contradiction as resources for learning. For instance, in reference to Cole's work on the Fifth Dimension, Gutiérrez *et al.* emphasize how "contradictions, experienced by us as conflicts" can be "a major source of change" [177, p. 217] and how diversity may be used "as a resource" to address program goals, provided participants are primed to see it as a resource. Said more explicitly, a CHAT perspective recognizes how the many frictions, breakdowns, and gaps present in classes —experienced often as "conflicts" —may instead be mined as resources for learning outcomes, rather than detriments to fully "solve" or overcome.

2.2.2 Constructionism and cultural constructionism

Computing education (alternatively called CS education) is a field arguably founded by (or at least first articulated by) Seymour Papert. His influence over the field continues to this day [12]; thus it is important to compare his theories of learning with others such as CHAT.

Influenced by Jean Piaget's constructivist learning theory, Papert developed a discovery-based learning pedagogy that he called "constructionism" and an associated educational programming environment called LOGO. The LOGO system allowed children to explore programming through (what Papert called) "body-syntonic" (embodied) reasoning with "turtles" that moved and

drew graphics on computer monitors in response to parametrized commands [314]. Papert's "constructionism" was a modification of Piaget's constructivism, which focused on individual learning processes through discovery, play, and exploration, instead of emphasizing any particular social or cultural underpinnings of activity. Unlike Piaget, Papert emphasized the influence of cultural artifacts in the learning process.⁵ He argued for the idea of "child as epistemologist" who "appropriate... materials they find about them, most saliently the models and metaphors suggested by the surrounding culture" [314, p. 19]. Here, culture is initially invoked to mean the tools, symbols, and materials provided to the child by their community. In this manner constructionism has a similar emphasis to CHAT on the role of artifacts and tools in the learning process; unlike CHAT, however, constructionism is relatively silent on social interactions among and between children. For Papert, where childrens' developmental deficiencies appear —say in understanding "permutations and combinations" in mathematics (p. 20) —these differences may sometimes be attributed to "our culture's relative poverty of materials" for playing around with those concepts (p. 7). He argues new computer-based tools can address this poverty.

Critical scholars in HCI often begin with a caricature of constructionism that assumes Papert believed children would best learn by themselves, away from teachers or adults [12, 80]. For instance, Chan argues that Papert "viewed children as innately able to teach themselves" and implies that that view led to the (failed) ideals of Negroponte's One Laptop Per Child project in how it overlooked teachers [80, p. 187]. Although Papert believes young children may "spontaneously" learn *very basic* concepts, such as counting or ordering objects

⁵"Where I am at variance with Piaget is in the role I attribute to the surrounding cultures as a source of these materials... I give more weight than [Piaget] does to the influence of the materials a particular culture provides" [314, p. 7, 20].

[314, p. 20], he states emphatically that “teaching without curriculum does not mean spontaneous, free-form classrooms or simply ‘leaving the child alone’ [but] supporting children as they build their own intellectual structures with materials drawn from the surrounding culture” [314, p. 30-1]. Clearly, then, Papert did not argue for classes free of mentors.⁶

In using the term “culture,” Papert denies that he is “trying to contrast New York with Chad [in Africa]”; rather, he is “interested in the difference between ‘precomputer cultures’ (whether in American cities or African tribes) and the ‘computer cultures’ that may develop everywhere in the next decades” [314, p. 20]. For Papert there is something about introducing the computer that alters culture, and the computer is sufficiently new that it does not matter what society, specifically, one introduces it to. The implication is that “precomputer” cultures are, in one central respect, impoverished, and that this deficiency in the tools, symbols, and materials of populations across the world may be resolved through the introduction of computational tools. Aside from a possible colonial impulse embedded in this assumption [12], Papert articulates throughout *Mindstorms* what he believes computer tools will provide: “[t]he intellectual environments offered to children by today’s cultures are poor in opportunities to bring about their *thinking about thinking* into the open...” (p. 28; *emph. added*). Thus Papert is concerned about *metacognition* (in the common sense definition of ‘thinking about thinking’), and in his view, all societies before the computer struggle to teach metacognition. These uses of the term “culture,” as Ames points out, risk “reduc[ing] the social world to a toolbox where children

⁶When I asked Mitchel Resnick and Natalie Rusk (the spiritual successors of Papert) at MIT about OLPC, they lamented the program, arguing that it didn’t embody constructionist principles and that they had sought to distance themselves from the project. Unfortunately, neither Chan nor Ames seems to have interviewed actual constructionists about their views on OLPC [80, 12]. This may lead to a biased feedback loop where artifacts [like OLPC laptops] “having politics” actually is the authors’ politics “having artifacts” in disguise [220].

might encounter gadgets that help them learn mathematics and logic, stripping away the complex *social* motivations and interactions that constitute culture” [12, *emph. added*]. Interestingly, Papert does not claim there is culture built into or embodied by the computer, stating emphatically that “[t]he computer is not a culture unto itself, but it can serve to advance very different cultural and philosophical outlooks” (p. 31). The computer —and, apparently, any peripherals required to communicate with one —is not “a” culture, but can embody a cultural perspective.

Papert’s positive framing of “computer culture” in *Mindstorms* appears inverted later on in Turkle and Papert’s paper on epistemological pluralism [407]. Turkle and Papert find that the “computer culture” has “discrimination... that is determined not by rules that keep people out but by ways of thinking that make them reluctant to join in,” citing how a female student felt side-lined in their computing classroom. Contrary to Papert’s position in *Mindstorms*, here the “computer culture” is no longer advanced by simply the introduction of computers, but rather refers solely to social dynamics outside of material factors: “[a]lthough the computer as an expressive medium supports epistemological pluralism, the computer culture often does not.” Here, the “computer culture” refers to the social world that the computer is embedded in; specifically, the practices and values instilled by teachers in CS courses that dictate how ‘best’ to program.

Ultimately, while there are inconsistencies in Papert’s arguments and definitions, there is less stark an opposition between cultural learning theories like CHAT and constructionism than is often assumed. And indeed although Vygostkyian sociocultural theories are often put in opposition to Piaget’s con-

structivism, Paul Cobb argues that the two perspectives are complementary in that each “tells half of a good story” [87, p. 17]: “Together they encompass the actively cognizant student, the local social situation of development, and the established... practices of the wider community” [88, p. 380]. Following the research of Paula K. Hooper —herself a student of Papert’s —we may refer to this reconciled perspective as *cultural constructionism* [203], aligning with the “cultural constructivist” perspective of Scott, Cole, and Engel [375]. Cultural constructionism emphasizes the role of the learner in constructing their knowledge through artifacts and technologies and their expression (and exploration) of their cultural identity/ies through this construction, while also theorizing the mediating activity system and adult influence as cultural and inseparable from cognition. This is important as the influence of sociocultural context, relationships and motivations in learning was arguably downplayed in early constructionist theory [203, 12]. The certain differences between constructionism and CHAT may therefore be the former’s emphasis on individualistic learning —on, let us say, a cognitive approach to learning —at the expense of social relations, and the latter’s emphasis on social learning, that may nonetheless underestimate or under-theorize individual learning.

Today, Papert’s ideas continue to exert a large influence on the field of K-12 CS education. What Papert originally meant by “computer culture” was perhaps succeeded by the terms *computational thinking*, introduced by Wing, and *computational learning*, developed by Papert’s spiritual successors Resnick and Brennan [63]. Computational thinking “involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science” [430]. Computational learning includes computational thinking, but reorganizes the concept into three aspects: “concepts,

practices, and perspectives” [63]. Concepts include the usual programming constructs of sequences, loops, events, conditionals, and operators, as well as data structures and pattern matching. Practices describe strategies students deploy for writing programs, such as iteration, modularization, testing and debugging, and reusing/remixing others’ code. Perspectives includes developing curiosity around how things in the world work, becoming producers rather than passive consumers of computing, and connecting with others through the development or publishing of computational artifacts such as games. Here, ‘perspectives’ appears like an attempt to bring social relations back into constructionism (although, for scholars like Kafai, it does not go far enough—she instead proposes the term ‘computational participation’ [224]).

2.2.3 Bennett’s Model of Intercultural Development

Emerging from Hall’s intercultural communication was a subfield specializing in how individuals become “competent” in interactions across difference [50]. Sociologist Milton Bennett’s “Developmental Model of Intercultural Sensitivity” specified six stages of development moving from ethnocentrism to ethnorelativity: Denial, Defense, Minimization, Acceptance, Adaptation, Integration:

1. **Denial** of cultural differences, where one’s worldview is considered equal to reality. Contact with outsiders may presume them subhuman.
2. **Defense** of one’s cultural worldview against outsiders, including denigration of differences or assumptions of superiority.

2.1. An alternative stage here is **Reversal**, which involves the extreme

denigration of one's own culture and assumed superiority of the outside culture. This for Bennett is also "defense," because those caught up in the reversal stage are defending a binary, oversimplified good/bad interpretation of the difference.

3. **Minimization** of cultural differences: Cultural difference is acknowledged but downplayed in favor of universal humanity. Similarities between cultures are overly emphasized.
4. **Acceptance** of cultural difference: differences are acknowledged and respected. This involves acceptance of differences in communication styles and non-verbal behavior, in addition to explicit differences like language or dress.
5. **Adaptation** to cultural difference: beyond acknowledging differences, one can empathize with the stranger's worldview such that they shift between worldviews (e.g., an American in Japan begins to bow and speak quietly in certain situations where in America they would be boisterous and loud).
6. **Integration** of cultural difference. Roughly speaking, the individual sees themselves as the other: they integrate multiple selves or identities (perhaps even calling themselves, say, "German" after a life in Germany despite living in the U.S. in their formative years).

Intercultural development augments CHAT and cultural constructionist theories to focus on how people learn about difference and how to navigate it—broadly, to 'think about their cultural thinking' [51]. In addition to learning about culture, it encompasses the making of friends across social differences, especially across societal divides and power hierarchies, and conflict resolution

skills. Intercultural development, I shall suggest, is a relevant mode of analysis not just for the present day, but for historical analysis as well: the degree of a person's intercultural development could determine what ways of being and knowing they supported or shunned.⁷ Although these stages are by no means absolute, I have found them to be extremely useful in characterizing and explaining behavior of not just children, but adults as well.

2.2.4 Critical Peace Education

Although this thesis does not focus on peace education in particular, the work of Zvi Bekerman has had an outstanding impact on my thoughts, and deserves an overview here. In brief, in international settings "peace education" has come under fire for doing nothing to change political dynamics that reproduce conflicts. Instead of imagining futures together, possibly by destabilizing existing power-laden social structures and group categories, peace education may unwittingly seek to maintain the order of things. After decades of work in Israeli-Palestinian schools that took an "intergroup dialogue" approach to peace-building, Bekerman realized that adults were (sometimes unwittingly) maintaining the social structure through group-making boundary work (for instance, how they separate the children during special events or respond emotionally to certain topics) [47]. Despite the good intentions of the school, children then learn these "tacit" or hidden cultural rules that reproduce the broader social divides by capitulating to their premises of difference. Unlike many people perhaps in education today, Bekerman became skeptical of emphases on

⁷For instance, perhaps Bertrand Russell's relative openness to East Asian cultures, among his other progressive stances, made him more amenable to Gottlob Frege's two-dimensional logic notation, which other European men of the time denounced or overlooked for how it jarred with their cultural conventions of the aesthetics of scientific practice.

social “identity,” arguing that “identity wins” over peace in classrooms where it is overly emphasized, or where adults are motivated to perpetuate their static notions of identity by projecting them onto children. Like infrastructure, here groups like Palestinians and Jewish people “are not a ‘what’ but a ‘when’ and a ‘how’” [46, p. 80]. For Bekerman, the solution was to “help our children become ingénues about the ways in which social categories are constructed and engineered by nation-states” and to look for ways to “sustain children’s cultures” especially when they conflicted with adults’ rigid, ossified ontologies of difference (p. 81). Real social change in a society, therefore, will necessarily cause discomfort to adults in that society.

2.3 Why “Intercultural” Computing?

Now that we have overviewed some key terms and frameworks, what do we call a computing education focused on issues of social and culturally-derived tension? Do we invoke “intergroup” —following the lead of intergroup dialogue programs —or, rather, “intercultural”? Or something else, such as “critical computing education” [409]?

To close this section, I argue for an “intercultural” framing to characterize my work going forward. For describing social dynamics in educational contexts, there are three points for choosing intercultural over intergroup. It suffices to provide an outline of the argument:

1. When deployed in analysis, “group” categories risk reifying boundaries of socially constructed difference by making the analytic mistake of using

the interventionists' categories to describe members' categories [68]. Especially in educational settings with children and adolescents, the usage of group "identity" can sometimes confuse "being" a member for "becoming" a member. This may lead to circular analysis. Unreflective application of group categories can then overlook cultural difference (including by otherwise well-meaning people), because they often suggest a bijection between group and culture categories.

2. In scholarship on equity and education, particularly the works of Lisa Delpit and Prudence L. Carter, tensions involving "race/ethnicity" in the U.S. are often code for, and obscured by, associated cultural differences [116, 77]. Thus, developing students' respect for differences is often framed by equity scholars in notions of "cultural flexibility" (advanced by Carter [77]) or intercultural learning, rather than intergroup learning, *even when racism is the salient issue, and even in the context of U.S. education.* (Equating race with culture is also a type of racecraft [142]; see Chapter 4.)
3. The generative definition of culture established by postcolonial and cultural-historical theories offers a rich theoretical grounding in materiality that is deficient in the intergroup literature. Activities, pedagogy, teachers, and technologies often embed and signal "group" boundaries by associations with cultural practices. Thus, culture is prior to group, in that the former generates the conditions for perception of the latter [45].

This argument would proceed by outlining work in, and criticism of, the terms "culture" and "group" across disciplines. These terms (and their relatives "race," "ethnicity," and "identity") have been contested in social science disciplines by social constructivists, post-structuralists, and other feminist and critical race theorists [74, 64, 199, 68, 45, 3]. Yet, though undeniably flawed, social

categorizing terms —as glosses for more complex phenomena —also perform valuable work for academics and practitioners alike [64, 94]. In opting for a generative definition of “culture,” rather than relying purely on group categories, I thus do not mean to avoid the term “intergroup” or using group’ categories, but rather to choose *culture* as the primary theoretical lens for my research.⁸ Per point 3 above, this also enables my research on the material aspect of coding (Chapters 5, 6) to integrate nicely with my educational work.

2.4 Conclusion

In this chapter, I reviewed several key terms, pedagogical and developmental frameworks that inform my work. At points throughout, I suggested that where one literature falls short, another may pick up the slack. In the end, I chose an intercultural framing, over an intergroup (or identity-based) framing. We shall return to questions of identity in more detail in Chapter 4, where I draw from the Fields’ book *Racecraft* to ground an activity for teaching the social construction of race.

⁸For instance, I may make reference to an “Hispanic American” group but my lens provokes me to investigate the cultural difference and heterogeneity behind this “group” as it relates to the populations under study; for instance, a Euro-American student could adopt elements of popular Hispanic American culture.

Part I

Programming and/in culture

CHAPTER 3
INTRO PROGRAMMING EDUCATION AS A SITE OF INTERCULTURAL
LEARNING

A growing movement is pushing for the integration of computer science (CS) into K-12 schools in the U.S. and beyond. Alongside widespread calls for equity and diversity in the tech sector, and propelled by government policy and funding decisions, computing education has become a prime site for the ongoing resolution of historical disparities in STEM fields defined along categories such as race, ethnicity, and gender [409]. Yet emerging work in diverse computing classes (and corporate settings) suggests challenges are often not resolved simply by bringing people together, whether in person or over the internet, even when programs appear to be focused on resolving disparities [13, 294, 366]. As the education scholar Prudence L. Carter remarks, “diversity is necessary but not sufficient for inclusion” [77]. Even when problems of access or participation to computer science are resolved, how can we address the “all too common” [366] sociocultural tensions that reproduce and entrench existing disparities?

These are not fundamentally new challenges, although they are relatively less explored in computing education research. Academics in intercultural and multicultural education, intergroup dialogue, peace education, and prejudice reduction (to name a few) have all studied problems of difference and tension, connected by the shared tissue of intergroup contact theory. Moreover, scholars in education researching K-12 classes and institutions have spent decades studying (and debating) equity pedagogy and policy (see, e.g., [116, 77]). What might the computing education field learn from this varied work? And how can these lessons and pedagogy be integrated into computing education by identify-

ing and leveraging alignments between computational and intercultural learning goals and pedagogy? Is there a role for computing in fostering “critical design experts,” as Bekerman and Zemblyas hope for in critical peace education [47]?

In this chapter, I explore how introductory computing education courses might support intercultural learning. My empirical work comprises two studies in Kenya and the U.S. The first study examines the Nairobi Play Project, an intro computing program for refugee youth in Nairobi and Kakuma refugee camp, Kenya, through ethnographic fieldwork across two program cycles. The second study examines social dynamics in a U.S. 6th grade classroom for five weeks in early 2020, until the onset of the COVID-19 pandemic disrupted proceedings. I describe the study contexts and methods, report findings, and then provide some overall takeaways.

3.1 Study Contexts, Programs and Methodologies

3.1.1 Nairobi Play Project

Program Design

The Nairobi Play Project (NPP) was designed as a progressive pan-African education model, theoretically grounded in critical pedagogy, constructionism [223], and intergroup contact theory.¹ The goal of the model is to cultivate intercultural learning between communities in or at risk of conflict. This iteration

¹This section was originally co-written with Ariam Mogos.

of the program was funded by The United Nations Children’s Fund (UNICEF) Kenya Country Program, and implemented by Kenya-based non-profit organization Xavier Project in coordination with the Nairobi Play Project team and local community-based organizations.

NPP was designed as 30 after-school sessions which run 5 days a week for 6 weeks, targeting 24 students and 2 informal educators per class. Each session is 2 hours long and generally starts with a warm-up activity or icebreaker, followed by a creative computing or game design session. The activities involve intercultural exercises, computational thinking and game design, and the curriculum strives to integrate these to cultivate problem-solving, self-expression and iterative practice. Programming activities occur in Scratch, a widely used graphical programming environment for early education [354]. Many activities are localized (e.g., redesigning the East African game *Mancala*). The program occurs in three sequential phases:

- Phase I of the program centers on building trust and friendship between students, introducing students to game design and storytelling, and learning basic computing concepts through remixing, debugging and making projects. This phase also involves a dozen warm-up activities, designed to enhance intercultural dialogue and teach computational thinking. Throughout Phase I, students are also asked to program in pairs; teachers were instructed to pair across nationality and tribe (although many paired across gender identity and made ad-hoc pairing decisions).
- In Phase II of the program, students decide on a community-based theme for a game they will build. After an introduction to real-world figures about the Sustainable Development Goals [296], participants work in their

teams to investigate and negotiate the who, what, where, when, and why of their game through discussion, peer interviews, and personal stories.

- In Phase III of the program, students create a fictional narrative using what they decided in Phase II to construct their game. Team members take on different responsibilities (coder, artist, writer), switch roles when prompted, and check in with each other until the project is complete. The program closes with a celebratory playtest.

While the core curriculum remains the same across classes, each teacher was instructed to interpret and adapt the curriculum according to their own understanding and in response to emerging local conditions and constraints. The five classes reported here were spread across two cycles (February-April and June-July, 2018) with a round of teacher professional development having occurred between the cycles.

Context

Refugee communities in East Africa are a diverse mix of cultures cutting across nationalities, tribes, languages, and religions. My Kenya-based study was conducted in the two geographies in which NPP operates: the more rural area of Kakuma refugee camp, and two urban communities of Nairobi. A total of 8 teachers and 232 students participated in NPP classes, split up into five sites (3 in Kakuma; 2 in Nairobi) and two cycles (120 students in Cycle 1; 112 in Cycle 2). All but two student participants were refugees. NPP recruited teachers through a partner NGO and paid stipends as allowed by the Kenyan government. While a few teachers dropped out due to repatriation, migration, or change in employ-

ment, all teachers appeared highly motivated by the intercultural component of the course and deemed it relevant to their lives.

In Kenya, many refugee children grow up and attend school entirely in refugee camps or communities [270, 280]. Pedagogy is often based on colonial-era educational models [148]: student-teacher ratios of 100:1 are common, and the goal is to pass standardized exams and obtain certificates to justify knowledge to employers [280]. Forms of creative problem-solving, indigenous knowledge, or intercultural understanding are typically neglected or discouraged. As one participant in the study summarized: *“In school, the teacher is writing here like this (points to blackboard), and we are trying to copy it [in our notebooks]. ... If you want to discuss, the teachers will not allow it.”*

Three of the classes I observed took place in Kakuma refugee camp, established in 1991 by the Kenyan government to house migrants fleeing from conflicts in Sudan, Ethiopia, and Somalia [73]. Located in the arid desert near Kenya’s northwest border with South Sudan and Uganda, today Kakuma² houses approximately 185,000 refugees and is one of two areas where refugees can live legally in Kenya [234]. Kakuma faces everyday conflicts from overcrowding, malnourishment, tribal differences, gender-based violence, and camp-host frictions [263, 104, 204].

I stayed at a compound in the nearby town of Kakuma (15 minutes’ drive away from Kakuma Camp 1), organized by a partner NGO. I reached the town by driving across around 100km of dirt roads from the nearest airport. The United Nations forbids non-residents from entering the camps after dusk, as it is dangerous; for instance, residents told me they avoided using backlit devices

²We refer to the refugee camp area as “Kakuma” throughout, but the camp area is distinct from the town of Kakuma.

at night because they worried the light would attract thieves. The sites were in isolated buildings behind fences, and youth were often seen playing football outside these areas at dusk. Quality food is scarce, consisting of UN-supplied rice and cabbage, chipati, small quantities of tomatoes and onions, pasta, eggs with discolored yolks, and often unidentified “meat,” with fruit being rare and costly. Of note is the large presence of the Turkana, the pastoral host community, which is in occasional conflict with the refugee population.

Although refugees have no legal status outside camps, many live and work in Nairobi, the site of the two other classes I observed. Life is tenuous and hard, as job opportunities are often low-paid and scarce; police corruption, xenophobia, and discrimination towards refugees is also widespread and growing [73, 323]. The refugees I encountered avoided taking lunch, only having a light breakfast and evening dinner. To support themselves or their families, students would take menial jobs, with boys mentioning construction work. Most students were in school, but a few were not and suggested that the class served as a distraction from realities at home. Classes were conducted in slum areas in both East and West Nairobi. These areas were both located beside marketplaces and were contained – as is common in Kenya – inside gated compounds with security officers present. At low traffic, both sites were at least 45 minutes away from the centrally-located, upscale gated community where I lived. For reference, a five-minute drive from one of these sites is an area where ethnic violence broke out in recent elections.

Class times were coordinated to occur after school. In Nairobi, teachers recruited students by posting sign-up sheets and posters in these areas, targeting a 1:1 gender ratio. Gender representation was balanced except for one class in

Cycle 2 which was majority male. Students came from areas around each site and transit costs were chief barriers to bringing in populations outside the local area. In Kakuma, teachers participated through refugee community-based organizations (CBOs). To recruit students, teachers spread the word by visiting nearby schools, coordinating with principals and school teachers, and putting up posters. NPP targeted a female-to-male student ratio of 3:2 in Kakuma and came close to reaching this target. Student nationality included Democratic Republic of Congo (DRC), Somalia, North and South Sudan (henceforth, S. and N. Sudan), Burundi, Rwanda, Tanzania, and Ethiopia, across varied tribal identities. Across all sites, literacy levels varied widely and most participants spoke at least two of three languages, to varying degrees of aptitude: English, Swahili, and a tribal language.

Methods and data sources

My investigation was qualitative, with data from a variety of sources: field-notes, semi-structured interviews with teachers and students, teacher self-reporting on a shared WhatsApp group, and pre- and post- surveys. For teachers, I asked for written consent, and interviews were audio-recorded. With the exception of Cycle 1 students and one Cycle 2 student, all student interviews were conducted on-site during class. I sought oral assent and hand-wrote all responses. Student interviews were in most cases conducted with a Swahili translator present, and some students switched between English and Swahili in their responses. Participation in the research component was voluntary, having no effect on students' ability to participate in activities. I offered small refreshment (tea, lunch, drinks, snacks) to participants when possible.

I gathered field-notes from the first eight days of Cycle 1 in Nairobi and professional development sessions in Nairobi and Kakuma. To preserve anonymity, we often refer to broad characteristics and use abbreviations: sites in Nairobi are N1, N2; sites in Kakuma are K1, K2, and K3. Students are labelled S1, S2, and so on, and teachers are labelled T1, T2, etc. Thematic analysis of coded fieldnotes during the first cycle provided grounding [82] for composing questions and observations for the second cycle. During Cycle 2 we gathered fieldnotes across all 5 classes, conducted semi-structured interviews with 8 teachers and 24 students (13F/13M; 13-19 years old), and held 4 informal follow-up interviews and phone calls with teachers. In addition, a teacher held brief informal interviews with 2 students. During Cycle 2, with the exception of N1, classes were composed of 24 students selected semi-randomly from recruitment lists of at least 48 students at each site. The randomization procedure stratified by nationality and gender to ensure diversity and reach gender targets. In addition to 302 pages of fieldnotes (164 handwritten, 138 typed in Arial 12pt. with standard margins), I took photos of documentary material and sketches of seating arrangements. Fieldnotes for Cycles 1 and 2 complemented one another by switching primarily from verbal to behavioral observations. In addition, teachers administered pre- and post- surveys. Although significant challenges hampered collection of all surveys, I include open-ended responses from 43 student's post-program surveys from four sites (24 K1; 9 K2; 9 N1; 1 N2) in my qualitative analysis. All data was analyzed through iterative thematic analysis, with a focus on intersections of intercultural learning with computing and pedagogy.

3.1.2 U.S. middle school study

Context and Program Design

The U.S. school study took place at a middle school in the northeast, hereafter called by a pseudonym, the Tangled Nest. Feeder schools included both elementary schools inside the county and rural areas surrounding it; racial demographics for the particular school are reported as 75% White, 10% Black, 7% Asian, 8% Other for the 2018-19 year. Students comprise a range of socioeconomic backgrounds, suggested by the feeder schools present, including rural, predominantly White working-class youth. These proportions are somewhat reflective of the racial/ethnic structure present in the larger U.S. society (e.g., in 2015 U.S. Census estimates, 73.3% of the population identifies as exclusively White; at the Tangled Nest, 75% of each class identifies as exclusively White). The school district that the Tangled Nest is a part of appeared relatively open to multicultural pedagogy and made active efforts to include “social justice” causes in programming; for instance, schools observed “Black Lives Matter” week during Feb. 4-8, 2019, and a 6th grade class at the Tangled Nest examined the effects of mass incarceration on youth.

The middle school year at Tangled Nest is broken into four 10-week periods. Two computing classes occur in each period, for a total of 8 classes. The initial plan for this study involved 2 phases. In Phase I, I aimed to carry out design-based research with ethnographic methods. Working with a teacher, I altered activities from the MyCS [145] curriculum (which the teacher knew well) to address intercultural goals, such as pair programming across difference or drawing each other. The beginning of Phase I also left time for designing activ-

ities concurrent with observations, adjusting and developing the curriculum as classes unfold, which is consistent with a CHAT approach that views emergent conflicts, breakdowns, or oversights as potential resources for learning goals [94]. The goal was to have at least one full period of iteration before Phase II, where building on my observations, I would implement a revised curriculum and run pre-post test and post-survey to measure changes in attitudes towards intercultural learning and any cross-group friendships that formed as a result of activities.

The research study at Tangled Nest was unfortunately disrupted by the the COVID-19 pandemic (and before that, a lengthy IRB process due to understaffing). Phase I of the study took place late January into March 2020 and was disrupted mid-way, with ethnographic notes and observations for 5 weeks instead of the full 10 weeks. Data included about four significantly modified activities, including a new activity type, interdependent programming, which emerged from observations of racialized inequities among pairs, particularly white males and girls of color. Though sparse, this data was enough to speculate on new avenues for research that aim to reconcile equity goals with intergroup contact in sites where dominant-grouped students have high levels of prior expertise. The study could not continue remotely due to the school districts' closure of research activity during COVID, and the fact that the teacher retired shortly thereafter due to frustration with the COVID situation.

Methods and data sources

For the U.S. study, I collected data through fieldnotes, select images and videos of student work, Scratch programs, and follow-up questions with students and

the teacher. In addition, I wrote analytic memos and reflections. We sought written consent from students and opt-out consent from parents with a letter that was sent home with students a week prior to the study. Unlike in the Kenyan study, I helped shape the activities and guidelines to align with intercultural goals, such as pairing students across groups and activities which involved exchange of interests or backgrounds. All fieldnotes were coded in the bottom-up, line-by-line procedure of grounded theory [82] with an emphasis on intergroup dynamics, and emergent themes were established through affinity diagramming. The codes and clusters of the U.S. study were then compared to the findings of the Kenyan study, with a focus on peer interaction. Common codes included: “budding bonds broken up,” “tensions over pairing,” “attending to intergroup friction,” “bonding over breakdown,” and “spatial sedimentation.” A U.S.-specific cluster is “white students centered” and a Kenya-specific code is “difficulties hearing” (due to noise pollution). Survey data from the Kenyan study was then compared to the resulting codes to identify common patterns or find points of divergence.

Key to my U.S. analysis is a shift towards racial “identification” that acknowledges how, in the terms of Ehlers and Piper, we are all passing [129, 335]. This challenging and divergent orientation seeks to be reflexive about its own reification of race and to surface who is doing the identifying [25, 134]. As I will explain in much more detail in Chapter 4, racial identities do not arise naturally; they are developed under various social pressures and surveillance, including researcher identifications, especially when participants are children [253, 189, 206]. To combat reification, some scholars suggest to put racial identifications in quotes [320, 106, 134]. Here I follow Fields & Fields [142] in using lowercase (e.g. “black,” “white,” “asian,” “hispanic”) to indicate a racial *identi-*

fication of students, and I use uppercase to signal an identity.³ In particular I use “of color” for students who were not identified as white and culturally Euro-American (i.e., a white Latina student would count as “of color”). Sometimes even when I know how students identify, I am purposely less specific to preserve anonymity, and use lowercase to surface the classifier-bundling work that I (rather than the student) am performing. Note that concepts of “race” are inherently unstable [264], and even under the rubric of identity, many researchers commonly shuffle students into monolithic racial categories despite additional complexity or student resistance (e.g., [128, p. 1579-80]). The experimental practice deployed here is meant to be uncomfortable and challenge readers. To be clear, however, I believe *both* identity/ification are important, and acknowledge that this flattening is a limitation of my U.S. methods, where I did not collect a pre-survey and was unable to follow-up with participants.

Finally, across my findings I often use the term “intergroup bonding,” and one might wonder how “intergroup bonding” is observed or measured. Salient indications of bonding emerge by paying attention to what students *do*, not just during activities but in the micro behaviors and choices students make during class transition periods and “gaps” in instruction [22]. These signals emerged from my data coding and included: partners sitting together unprompted at the start of class after they were paired together deliberately on a prior day; switching the mouse regularly during pair activities; positive body language (such as feet turned towards a partner [297]); asking a group member for permission or feedback; helping a partner with a non-computing task; who students exited class with; and the presence of shared laughter or giggling (social laughter trig-

³We acknowledge the importance of capitalizing “Black” when referring to ethnic Black American or diasporic identity. Our goal is rather to surface power in (imposed) identification. This acknowledges that not all students perceived as Black may identify as such, although people in the U.S. often assume they will [206, 253]. It also acknowledges our study limitations.

gers powerful bonding agents in the brain [269]). The absence or opposite of these signals could indicate a negative outcome –for example, pairs splitting up as soon as the activity ends or students sitting at a distance from others. I append the term “intergroup” to remind readers that each phenomenon occurred over interactions across *what adults perceive as* a contextually-significant difference. Finally, when I use the term “preparatory privilege,” I mean students who signalled prior knowledge –i.e., whether by correcting the teacher about an obscure Scratch feature, announcing that they knew Scratch beforehand, bringing up a game they’d programmed prior to the class, etc.

3.1.3 Positionality and Collaborators

The researcher and author of this thesis identifies a Euro-American male. He is racialized as white in the U.S. and *mzungu* in Kenya, and students may react to his presence in classrooms in significant ways that impact study observations. In Kenya, differences in language and power dynamics could affect understanding and interaction; he often followed up with students and teachers to double-check emerging themes. In both computing contexts he may be viewed as having authority by students by virtue of his racialization and gender.

For the U.S. study, he had prior knowledge of attending a public middle school in a similar diverse socioeconomic region in the East Coast. The middle school he attended had similar racial/ethnic make-up as the Tangled Nest, although the divide was probably starker in the former, due to tracking. He was surprised to recognize some of the same dynamics playing out in the Tangled Nest, with some of the same popular culture like manga, Pokemon, Naruto,

Battle Royale, and even the old cartoon Rugrats being popular among students.

This work could not have been carried out without several collaborators and advisors. Most importantly I am indebted to Ariam Mogos, who founded and designed NPP. Ariam is an Eritrean-American woman who has extensive experience designing and delivering creative computing programs around the world, including in Sierra Leone, as well as negotiating with international partners. Professors Kentaro Toyama, at U. Michigan, and Steve Jackson, at Cornell, served as advisors for the NPP ethnographic project, whether in analyzing field-notes or offering day-to-day support, and I am indebted to their attention and care. Professor Tapan Parikh offered support and feedback especially during the U.S. study portion of the project, and the resulting TOCE paper. Together, these three contributors are faculty members at U.S. universities who collectively have several decades of experience in HCI-for-development research in South Asia and sub-Saharan Africa. Of all of us, however, only Ariam speaks one of the various non-English languages spoken by NPP students (Amharic) and has lived experience with any of the cultures represented in the communities of study in NPP (not to mention experience as a woman of color in STEM). These positionalities posed practical and ethical challenges during the work.

3.2 Findings from the Kenya Study

Here I recount findings from the Kenya-based study. Findings are categorized into cases where (1) computing activities appeared to support intercultural learning, (2) obstacles emerged to such learning that had to be managed or overcome, and (3) moral dilemmas complicated normative accounts of intercultural

outcomes. Note that while teaching computational thinking and programming skills were key goals of the program, my focus here is squarely on intercultural learning.

3.2.1 Computing Education in Support of Intercultural Learning

A consistent finding across our sites was that computers provided a powerful incentive to gather across divides. Students often cited computers as the sole reason they took the course, voicing a vague belief that computers will help them in “the future” though many could not elaborate exactly why. Some took pride in being the first in their family to use a computer, e.g., *“Our father, our grandfather, our tribes – they have never touched a computer.”* Others emphasized a desire for credentials as printed certificates. Both aspects seemed to justify the participation of students to parents. T8 appealed to aspirations: *“You have to [tell parents] the advantage of [the computer]... ‘The future we are going in, it is all about technology.’”* In particular, the presence of computers could mitigate mother’s concerns about their daughter’s continued participation. Somali girl S24 said *“My mother is okay with [the class] because I am the first person in the family using a computer.”*

Once students were in the door, the question became how to engage them in the program’s intercultural goals while satisfying their appetite for computing activities. By design, NPP blurred the distinction between the two goals: activities sometimes had simultaneous intercultural and computational learning objectives. Both high tech games and low tech activities could put students

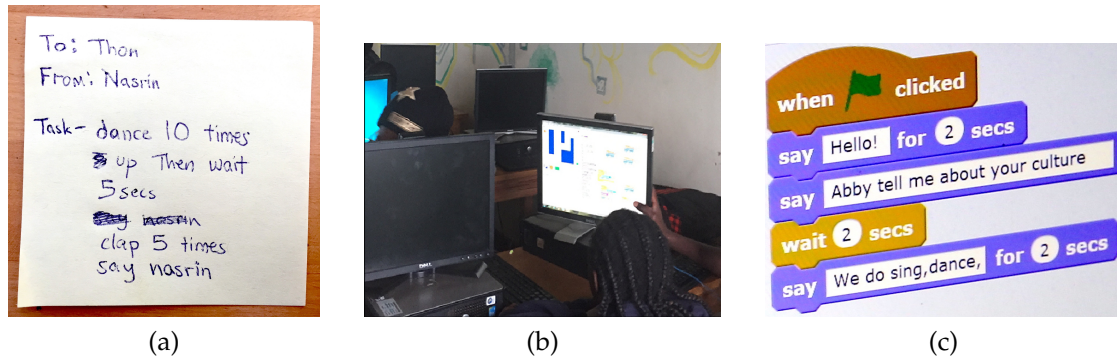


Figure 3.1: Interactions over computing activities: (a) During a warm-up, a Somali girl instructs a Dinka boy to perform a procedure (reproduced, with names altered); (b) One cross-cultural pair shares code to another pair by rotating their monitor; (c) Coding part of a conversation about culture between characters.

in the shoes of another. For instance, “Empathy Notes” – labelled an intercultural activity – asked students to write instructions to a fellow student which that student then has to perform; these looked like pseudo-code (Fig. 3.1a). Unspecified aspects of pseudo-code could surface culture; for example, a Somali girl was instructed to sing, and proceeded to cover her face with her chador and sing softly in Somali. Later discussion addressed how students felt writing these notes. Many participants attributed later cross-cultural bonding to similar low-tech warm-ups and their role in sanctioning intercultural interaction.

Mechanisms of bonding often seemed to revolve around *shared humour* arising from frictions between participant expectations and outcomes. While such moments could be intentional (S13: “*sometimes, we create games that are funny*”), even in cases of communication difficulty, unintentional humour seemed to play a more prominent role. Multiple teachers and students attributed the cause of laughter to “mistakes” or incongruities (e.g., a mouse chasing a cat), backing up fieldnote observations (T5: “*Once they find a mistake or [do] something amusing, they laugh, and sometimes they clap their hands.*”). Breaking the silence between them, a Congolese boy and a S. Sudanese boy started giggling as their cartoon

cat became stuck in a maze wall: *“We want to move it down but it moves up,”* said one. Breakdowns over low tech activities had similar effects; for instance, in one activity two teams gave each other “code” to act out, resulting in laughter when students made a mistake. The same activity seemed to prompt students to open up and speak with one another.

Other moments of bonding appeared to be driven by limitations in resources and infrastructure. Individualized devices such as the mouse, keyboard, and earbuds presented bottlenecks that produced frictions between pairs negotiating for use. Students had to vie for control of a device, providing opportunities for assertion and restraint, and for teachers to encourage active communication. When asked how he made cross-cultural friends, S17 replied:

Our teacher told us you must sit together – for example you’re Congolese, you’re Sudanese. They mix us... We have to communicate. Because there is only one computer. You cannot make something without the computer.

Communication could occur through nonverbal means, including in cases of communication difficulty, but even when language was not an issue. For instance, S6, a Congolese boy, said he was having difficulty conversing with S27, a S. Sudanese boy, even though they were both fluent in English. *“If there was a mistake [my partner] doesn’t speak, but he shows (gestures as if pointing to the screen) – shows the mistake. And he took the mouse [from me] and corrects the mistake.”* Indicative of intercultural competence [113], S6 asked S27 to modify his behavior: *“His voice is low, is low!... If you say to him to speak loudly, he will at that time. But not normally.”*

Instances of pair bonding over a shared task could bridge what appeared

to be even large divides. Several teachers began to see Somali girls in particular *“sitting with boys from different tribes.”* T4 regarded this as *“very rare”* in Kakuma camp or even *“impossible,”* as Somali girls traditionally only engage in limited forms of communication with boys. In one case, a friendship developed between Somali girl S26 and S. Sudanese (Dinka) boy S25, who went to a boys-only school. Early in the program, the girl communicated mainly with her Somali friend. Yet in a later interview, the boy said he was making friends with both girls, citing his partner’s abilities: *“I feel good because the girl is very social... She really understands. I like the way [she] concentrates... It really helped me.”* The girl initially hesitated to reciprocate. When given a chance to choose a partner from the class, she didn’t choose her partner *“because he doesn’t understand that he’s a boy.”* Yet two weeks after the program ended, she wrote, *“yes, I have a new friend [S25] who was my computermate and become my best friend.”*

Yet sites of bonding, particularly where prompted by breakdowns in understanding, were not just limited to deliberate pairings. We observed many instances of brief, one-shot interactions across pairs or in unstructured moments. For instance, a Burundian boy and a Ugandan girl demonstrated their code for another pair –a Burundian girl and a Somali girl. This kind of helping often entailed rotating the screen or laptop so that others could see, as in (Fig. 3.1b). In other cases, students bonded during gaps in instruction; for example, while teachers prepared, a Kenyan boy explained the concept of a ‘forever’ loop to a Rwandan girl absent from the previous class.

Finally, many moments of friction occurred over group projects, where students had to contribute and codify their ideas. Such moments included: ideation and brainstorming; constructing scenes relating to culture inside

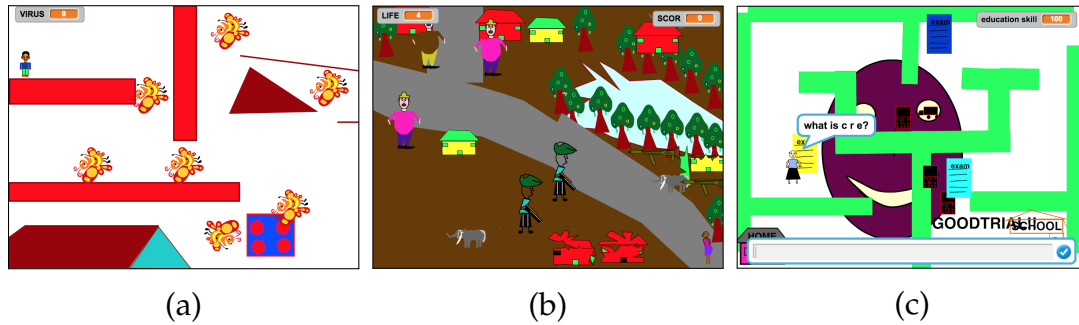


Figure 3.2: Group game designs embody experiences shared across difference: (a) a boy avoids mosquitoes to reach red malaria pills; (b) a woman flees across a border as “terrorists” block her path; (c) a student vies to finish their education by answering questions on exams and dodging distracting mobile phones.

Scratch (Fig. 3.1c); playful argumentation with hands during prototyping; and cultural conflicts over design choices. For the last, a team of four (Burundian (M), Somali (M), Dinka (2 F)) decided on “malaria” as their final game topic, and had a disagreement whereby Dinka members argued that reaching an indigenous plant medicine should be the goal. Somali boy S22 disagreed, saying that was “*old fashioned, you should embrace the new one, you shouldn't believe that anymore,*” and argued for red malaria pills as the goal. The team went with S22’s design (Fig. 3.2a).⁴ Final game topics included peer pressure to smoke bhang (marijuana); the government blocking UNHCR’s attempts to expose corruption; cleaning up Uhuru park; avoiding “*bad boys*” on the street; planting trees; the importance of listening; and the problem of open defecation. Many themes reflect shared experiences – such as migration to Kenya or school education (Fig. 3.2) – and thus indicate design being used as an opportunity for building common ground. Others, e.g. a game about child brides, reflect possibly more personal or traumatic experiences of their group leaders.

⁴Although this reflects how cultural difference can surface from directed pedagogy, it also represents a potential complication to intercultural efforts that we will explore in a following section.

3.2.2 Overcoming Obstacles to Intercultural Learning

As already intimated above, the intercultural objectives of NPP also faced numerous practical obstacles. Some of these emerged from gaps in program structure and design that were filled in by caring initiatives taken by teachers and students [229]. In these cases, built-in diversity could be used “as a resource” to contribute to intercultural goals, *provided they were consciously attended to*.

As noted above, aspirations around computing served as a key motivation for gathering and participation. Once students arrived and learned who was in the room, however, there was no guarantee they would stay. One of the most prevalent cases of friction occurred between Dinka and Nuer participants, communities in active conflict.⁵ A Nuer teacher encountered two Dinka students in his class who stopped coming upon learning his tribe. In response and on his own initiative he traveled to a Dinka section of Kakuma to encourage the boys to remain in the program. “*Even my people say, ‘Why are you going there? They’ll kill you...’ No, they won’t kill me. I’ll talk to them.*” The boys returned after his visit and finished the program, indicating to him that their negative perceptions had shifted, albeit tentatively.

Even if students were in class and eager to listen, they often resisted intercultural interaction. Across sites, students’ natural tendencies were to sit with others like them. This resulted in groupings by nationality, tribe, or school early in classes. For instance, we saw Somali girls clustered into one corner; students clumped by the same school uniforms; and the only three Congolese students in one class huddled around one laptop. These tendencies to sit together (or

⁵A 2018 report funded by the U.S. Institute of Peace estimated approximately 382,000 casualties in South Sudan between 2013-18, resulting in an influx of 2.5 million new refugees in neighboring regions [84].

apart) could crystallize through force of habit, limiting space to maneuver, and the reinforcing materiality of devices. At some sites students labeled their laptop boxes with their names or grew attached to project files saved on laptops; at classes with more computers, some students were seen working alone to take advantage of additional resources. Devices could distract students (T4: *"They were eager to use computers... I tried to explain them 'no, this [intercultural activity] will assist you in your lives."*), which could suppress encounters especially during gaps in instruction. At all sites where the internet was present, students visited YouTube (in some cases, wearing headphones and shutting others out) or played video games – media which could include violence. By contrast, in sites without internet, students were seen collaborating and focused on programming projects, especially before classes began.

Yet these situations presented opportunities for teachers to make intercultural goals explicit. By breaking up order and ritual, the *act* of disruption itself seemed to help students to reflect on their mindsets. S13 said, *"You cannot stay in the same place without interacting with each other. The teacher would tell us to communicate with each other, to get to know each other better."* Moreover, the reinforcing materiality of devices also provided powerful opportunities to have the opposite effect, to act as glue to sustain otherwise contentious (or nascent) pairings. Indications were that the intercultural friendship mentioned earlier between S. Sudanese boy S25 and Somali girl S26 only occurred because these students sat together over a sustained period, tied materially by their labelled devices after paired by the teacher. In other cases, students who were paired on previous days could return to those locations by their own initiative, suggesting that the "sticky" and affective properties of devices might be employed to sustain new arrangements.

Language differences could also hamper communication. Although NPP did not enforce a language of instruction, lesson plans were written in English and teachers chose to teach in English (even at sites where Swahili was common), switching into Swahili only occasionally. T3 summarized their position along three points: English is the “*language of computers*” (OS and Scratch software were both in English), Swahili does not have well-known terms for many computer peripherals or topics, and many students believed that if they learn English, they will be more successful in life. This gatekeeping proved difficult for Swahili speakers less versed in English, to the extent of causing dropout in two captured cases. Yet these frictions and oversights could also be fuel for interaction. Students used these challenges as opportunities to help others through translation, with some noting they were making cross-cultural friends through such transactions. For Sudanese students, English knowledge seemed especially advantageous. While working together on the final game project, S. Sudanese boy S10 and Congolese boy S11 both cited S10’s knowledge of English as a source of bonding: “*When English words become difficult for me, [S10] will come and help,*” said S11.

Still some opportunities for dialogue may be inhibited from perceptions of appropriate cultural behavior. Teachers used this resistance as a chance to encourage these students. T5 recounted a Somali girl who returned to the centre after the program ended to continue programming practice:

First of all, she’s very shy. She doesn’t understand, she doesn’t ask questions... I asked her, “Is that how you behave at home? Do you speak with your father and brothers?” She says, “No, I’m used to talking to them.” I said, “Here, we are united by this class regardless of our culture and reli-

gion." Then from there... I made her the group leader so that she'll stand up and read what they discussed. From there, she gained confidence... In her religion, she says that a woman can't stand in front of people. I say, "No, how come? I've seen Somali girls who speak and stand..." So, in the end, she has changed.

It is important to note that the ability to turn challenges into resources for intercultural goals hinged upon the built-in diversity of classes. At N2, logistical challenges prevented NPP from selecting students, resulting in a student composition mostly from the same tribe. The class resulted in a reversal of intercultural goals: one boy of a darker skin tone and different nationality than the majority appeared to be making new friends in N2, even while not making friends at school, yet his post-survey responses suggest *negative* effects on his attitudes towards cultural others. It turns out that he had experienced instances of exclusion: His teacher overheard his teammates insisting on using their language saying, *"hey, when you speak your [tribal] language, do we understand?"* The teacher thought that the majority of students at N2 were not internalizing the cultural activities: *"It's like, they're not willing to go past their culture."* This case highlights a potential crux, that class make-up must be diverse enough that no single group stands out as the majority.

3.2.3 Complications of Intercultural Learning

The previous sections involve instances of intercultural learning, particularly at sites of frictions, breakdowns, and gaps. However, whether instances of intercultural engagement are truly positive is not always clear. We now turn

to more complicated examples of intercultural outcomes, representing value judgements on meaning making and behavior change. These examples illustrate how frictions and gaps, before framed as opportunities for learning, could also shift into ambiguous or negative outcomes even when participants are following structure or bonding across difference.

First, while the need to communicate over shared devices had the potential to bridge divides, it could also be a site for conflict and inequity (as others have cautioned [324, 255, 357]). In one case, when his partner took complete control of the computer, a boy left the pairing to work with a girl from his own tribe, remaining with her for the remainder of class. At the same time, students were aware of such dynamics. S21, a Ugandan girl, liked to work with the Burundian boy she was partnered with saying that “[he] is ready to listen... everything is equal,” but “when I observe other pairs, one person is taking control of the computer.”

More problematically, even if all learning outcomes are ostensibly being met, with students designing self-expressive games and bonding with those from other cultures – *what* students are bonding over could be controversial. For example, intercultural bonding can occur over shared xenophobia. In one case, a team designed a game whereby migrants reclaim lands by killing all male members of another tribe. The team was composed of three boys – two Congolese (from different tribes) and one S. Sudanese. One of the students, who came from a tribe recently displaced due to violent conflict, explained that he wanted to educate refugees on how “to emerge from struggle and suffering.” He said his teammates could identify with the game’s story. We pressed him several times as to whether there was another solution or design, but he finally explained, “If you don’t kill the thieves, they will come back. If you kill them, the story is finished

because they are dead.”

Third, there were instances of intercultural learning which appeared to bring pressures to assimilate to the dominant culture. Before, we showed how gaps in the program provided space for dialogue; but they could also present opportunities and/or pressures for students to practice new behaviors typically prohibited by their culture. One prominent instance involved Muslim (Somali or N. Sudanese) participants. These students wore hijabs or chadors and came from families that prohibited physical contact – including handshakes in greeting – with males outside their family. Teachers told us that this practice initially created negative perceptions among participants from Great Lakes cultures. At multiple sites, participants negotiated this practice during a warm-up activity that involved designing a new handshake. Though not forced to shake hands by the teachers, some Somali girls began shaking hands from beneath their chadors; others eventually with their bare hands. These students were seen shaking hands during unstructured moments days or weeks later, such as after peer game demos or before class.

3.3 Findings from the U.S. Study

A follow-up U.S. study took place in a 6th grade classroom across 3 periods taught by the same teacher, Mr. M. I participated in the classrooms as an observer for five weeks, until the onset of the COVID-19 pandemic. Some of the findings were similar to the Kenyan study, such as how deliberate pairings crystallized over multiple days. For example, I observed bonding over breakdown and humor frequently, as well as students “teasing” each other through the cre-



Figure 3.3: During a pair programming activity, a hispanic girl and a white boy create a game about helping an old woman cross the street. The scene ends with the old woman thanking the man and the man replying “vale Boomer” (“OK Boomer” in Spanish). The pair were observed giggling together over the project, but the U.S. teacher, an older male near the boomer generation, reacted negatively to the epithet.

ation of media (e.g., in an activity of drawing one’s partner, a black girl drew her white, female partner with big red lips as a joke). More unique findings, however, emerged from the differences between contexts, especially the preparatory privilege (prior knowledge) held by some Euro-American male students. Here I report on the U.S. study, focusing on some of the more unique findings for this context.

3.3.1 Intergroup bonding over othering and stereotyping

Similar to the Kenyan study, sometimes intergroup bonding could occur around instances of othering, including humor around stereotypes or epithets. For instance, a white boy and a hispanic girl (who had recently immigrated from South America) bonded over the creation of a game about helping an old, pale-skinned woman cross the street; without any direct instruction to do so, students chose in-game dialogue to be in Spanish. Using Google Translate to discuss and regularly passing the laptop between them, they were later observed

giggling over humorous breakdowns and purposeful choices, such as the dark-skinned male sprite picking up the old woman and carrying her across the street. The researcher congratulated their efforts and attended to other pairs. However, upon viewing their work, the teacher found the game included the phrase “vale Boomer” (“OK Boomer” in Spanish) that he thought disparaged older generations of which he was a part (Fig. 3.3). He scolded the pair for their choice and encouraged them to remove the epithet. This appeared to demotivate the pair the next day, with the boy hiding his face in his arms and adopting an oppositional attitude. In this choice, the teacher’s resolution of friction may have negated their social bonding by problematizing the outcome of their interaction.⁶

In informal interactions, people can “play” with group stereotypes as a bonding mechanism; in this type of humor, the friction of stereotypes can “open shared understandings of the underlying assumptions of dominant frames” in order to “destabilize them through making those assumptions visible, and laughable” [445]. As this example illustrates, the line between what is truly othering and what brings people together is not always clear. Teachers must balance whether to condemn prejudice or stereotypes expressed towards others (e.g., older generations or ethnic groups) with the potential benefits of acknowledging and supporting –rather than shutting down –bonding that occurs around it. Such situations also express complications with projecting researchers’ and teacher’s sense of ethics and political correctness onto students [388].

⁶“OK Boomer” has been used by young adults and teenagers, particularly those marginalized by U.S. society, to critique the attitudes of the (particularly white) baby boomer generation; however, we acknowledge that this phrase also exists in a context of rising ageism worldwide [277]. The teacher’s reaction to the essentializing phrase may be a combination of his socioeconomic status and prior challenges to his authority in other interactions with the white student in the pair.

3.3.2 Intergroup bonding with unequal control

As scholars have shown for U.S. contexts [252], power dynamics can manifest in unequal group negotiations where dominant cultural practices were centered. Students privileged by the society could also come to class with prior knowledge of computing [271]. I observed preparatory privilege strongly in U.S. classes, although more along the lines of race than gender (some white or asian girls in 6th grade classes seemed just as experienced and interested in coding, which could reflect either the young age group or the shifting dynamics of early 2020). White boys would express prior knowledge of Scratch –in a few cases even correcting the teacher during a lecture –that I perceived as impacting their interactions with other students, such as dominating control of the computer during pair programming activities.

Prior work suggests that preparatory privilege and in-grouping in nearly-all-white classrooms, and the ostracization that can result, is a key reason why students of color elect not to participate in CS courses [294]. This implies that white students actively express prejudice or avoid interacting with students of color. Yet I found that negative dynamics could re-emerge *even when students were ostensibly getting along and bonding across difference*. In other words, in some intergroup pairings, bonding appeared to occur while partners of color are relatively less engaged in activities and white students are centered through preparatory privilege and/or assumptions of dominance [271].

A common case was a white student who controlled the computer for almost the duration of a pair programming activity with a girl of color and where the pair were otherwise seen laughing, bantering or smiling together. Here is one example of these dynamics during a two-day Scratch activity:

Kemi (black) and Jeff (white) are giggling over their screen while Jeff controls the computer. I come by and ask, "What's the scenario?" Kemi looks up and tells me, "We're fine." Struggling to do something, Jeff rotates the male sprite upside down. They both laugh. Jeff clicks something and Kemi laughs harder: the man is now laying on his side in the street... Later in the class, Jeff is slamming his finger into the Chromebook touchpad on the stop sign (likely because the laptops are slow). "We need more time cause we're laughing," Kemi tells me. The next day, Kemi has her knees on her chair, her feet oriented towards Jeff with his blue hood up. They are again chatting as Jeff controls their laptop.

As this incident suggests, a black student may signal that they believe computing is not "for" them even while bonding with a white student around a computing task. Their collaboration is still "around" the making of computational artifacts, but much less "through" the making of them. Following the path-breaking work of Paula K. Hooper, I remind readers how racist messages from the media, school and peers, as well as lack of representation and invisibilized histories of Black American scientists [276], often signal to students of color that computing is not for them [271, 26, 333, 203]. The student with preparatory privilege may also make it easier to spur or encode humorous scenarios, since they are likely to experience less frustration while programming. Importantly, we do not mean to imply the quality of bonding was especially deep or something Kemi would have chosen to do absent the school context; however, the dynamic could emerge.

Not all interactions where a white student bonds with a student of color are negative, nor does helping require forcefully taking control of a computer. For

instance, in one exceptional case a white boy, Mark, and a black girl, Yasmine, appeared to be prior friends, sitting near one another, bantering, and teasing each other. Like some white males in the course, Mark demonstrated high competency with Scratch –but so did Yasmine, who often dominated control of the computer when paired with other students, including white students. In one exchange around movement for a sprite, Mark calls the researcher over (a common occurrence):

“Can you show her how to not glide [use move blocks instead of glide blocks]?” As I approach, April, a white girl across from the pair, turns her laptop around and tells Yasmine, “See you should do it like this.” “See watch,” says Mark, clicking the arrow keys on April’s computer. Yasmine says, “I want to do that help me.” She turns her laptop towards Mark. As he makes changes, Mark teases Yasmine over not knowing the right block to use.

Power dynamics emerge in this interaction, but Yasmine also signals her interest –“I want to do that” –and her agency in commanding Mark to show her how. Yasmine’s granting of permission to Mark is different than Mark taking control of her computer unasked. In other interactions, Yasmine was paired with April, and generally directed the interaction.

3.3.3 Teacher discomfort around breaking up order

One of the key design strategies of fostering intergroup bonding is to mix students. But to break up social order, teachers must make deliberate choices and

commands. These expressions of power often face resistance by students when grouping across gender and ethnicity or breaking up friendships. They also require potential trade-offs between intercultural and computational learning goals that teachers must balance.

In the Kenyan study, teachers expressed few reservations at breaking up order and countering student resistance. Teachers often cited their own learning to interact across difference as their motivation. However, in the U.S., the white male teacher expressed discomfort with deliberately pairing students of color with white students or breaking up prior friendships. To distance himself from the process, he passed out numbered popsicle sticks to randomize students and pointed to places in the room, calling out numbers. This method had a chance to pair those who were already friends.

I emphasize that prior intragroup friendships did not automatically mean distraction, but a few friendships did appear to distract students. Reacting to these dynamics, in a later class Mr. M paired students deliberately. Two black girls resisted:

Mr. M is telling students where to sit when they come in, stopping them and pointing to tables. "I need you to sit there," he says. Tiana, a black girl, spins around. "Why?" "Sit there," says Mr. M, pointing to the bottom-left table. As more students shuffle in, Mr. M. directs Alyssa, a black girl whom Tiana is prior friends with, to the back-right table... "Oh my God," Alyssa says, moving to the seat. Other students seem upset about the assigned seats, and the class is noisy. Mr. M asks the class to settle down. Alyssa raises her hand and Mr. M calls on her. "How come we have assigned seats?" The teacher sighs. "Because I needed to have people away

from their friends today.” Alyssa, who had stuck by Tiana for the last two activities, sighs dramatically. Mr. M says something about being honest.

On a prior day, Tiana was observed positively collaborating with a white girl, while when paired with Alyssa could be distracted by conversation. After this split, Tiana engrossed herself in designing a Hero’s Journey maze for her dance teacher, drawing a stick figure with long wavy hair surrounded by students; later she asked me for feedback. Meanwhile, Mr M. attended to Alyssa and the white girl beside her, who were later seen laughing together while leaving class.

This example surfaces the tension between breaking up intragroup friendships for marginalized youth and intercultural goals. Yet the example also illustrates a further point: teachers do not simply exercise power in groupings, they often have to provide *reasoning* for why. This is especially important for marginalized youth, who learn to be suspicious of authority figures and assert agency through resistance [367, 320]. Mr M.’s reasoning for disrupting social order (first ignoring Tiana’s “*why?*” and then, “*I needed to have people away from their friends*”) expressed a desire to avoid conflict and obscure the reasoning, which is different than reasons given by teachers in Kenya who were explicit about their (and the program’s) intentions for intergroup bonding.⁷

I also observed prior intergroup friendships in classrooms, such as white girls being prior friends with girls of color. The U.S. teacher responded to prior intergroup friendships in various ways. In some cases, he let prior friends re-

⁷For instance, when asked how they countered resistance, a Nairobi teacher recalled: “*We told them the importance of team work and helping each other to learn... I remember we did it by force sometimes, usually saying, ‘You have to stay with this person.’ But like... [even when] forcing the person, telling the person the importance of that.*” Even when one teacher gave up splitting students, he “*started showing them why we are doing the warm-up activities*” and claimed the students sat of their own accord with unlike others later into the class.

main together, viewing them as occasions to support, rather than disrupt. Typically, however, prior intergroup friends were broken up over randomization techniques for pairing. In these cases, the method and scaffolding of grouping students becomes important –randomization can break up existing intergroup friendships, but separating existing friendships during shorter activities, especially early on, may contribute to network bonding effects.

Comparing the Kenyan and U.S. findings, we arrive at two concerns when grouping students: *how much* and for *how long*. On the one hand, regularly mixing students seems to have positive effects: some students met and learned they got along well together, while other pairs that did not work out (whether because of control inequities or just differences in personality) could feel secure in the knowledge that it was temporary. On the other hand, mixing up groupings too much could break up intergroup pairs who seemed to get along well, disrupting the potential for deeper bonding. For instance, a white boy randomly paired with a hispanic girl told to the teacher, unprompted: “*I thought it was fun –working with someone I couldn’t communicate [as well] with.*” The pair would then come into class and sit together, but gradually drifted away from each other after split up by random pairings.

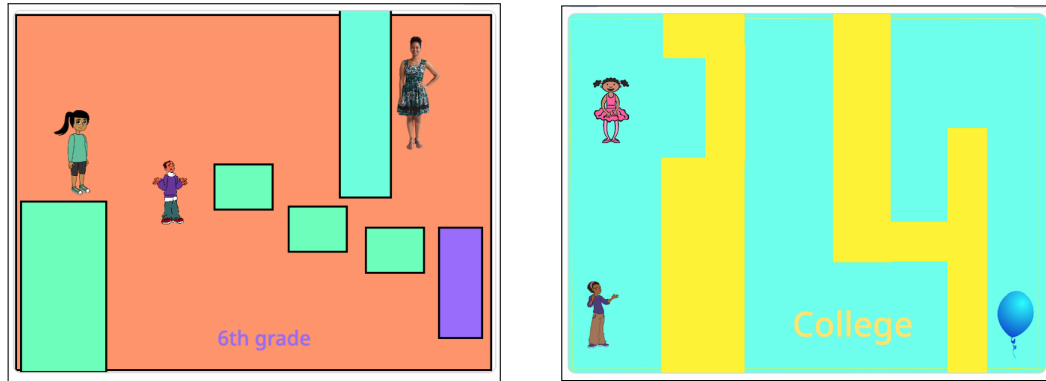
I speculate that a strategy of early mixing, followed by deliberate, longer-term pairings by an attentive teacher (who observes what students seem to get along, which hold certain social capital, which have social anxiety, etc.) may be better at fostering positive intergroup interaction. To do this, however, teachers must be willing to exercise commands in classes and utilize their social intuition when grouping students. While these expressions of power may be uncomfortable and error-prone, the Kenyan and U.S. studies show that countering

students' resistance to pairing could result in bonding that may otherwise not have occurred, especially when sustained over multiple weeks.

3.3.4 “Interdependent” programming

Consistent with prior U.S.-based work [271], I observed tensions between students' prior knowledge which fell along racialized and gendered disparities in the society. In the U.S., this divide was particularly prevalent among white boys and girls of color, where some white boys came to class with high levels of prior knowledge and in pair activities dominated control of the computer. Above differences in prior knowledge, this assumption of dominance and centrality is a manifestation of whiteness [345]: at very young ages, children in the U.S. learn to be biased towards whiteness, behaviors and beliefs entrenched by social segregation and cultural representation [394].

After observing this type of dynamic in the first three weeks of the study, I designed a new type of activity called “interdependent programming.” In an interdependent programming activity, students are grouped and work independently (on their own laptops) but the activity requires active sharing of information between collaborators. This is distinct from, though similar to, a ‘jigsaw’ or ‘hybrid’ method where each student in a diverse group is responsible for separate pieces of a larger project [313, 441]. The teacher and I tried two different activities, “Drawing One Another” (where pairs drew each other to learn drawing in Scratch) and “My Partner’s Hero’s Journey” (where pairs coded each other’s maze games around a personal hero). Both were modifications of MyCS [145] activities and the latter similar to a maze activity from the



(a) Jordan's game about Evan's hero, his teacher. (b) Evan's game about Jordan's hero, her mother.

Figure 3.4: Two partners –a black girl, Jordan and a white boy, Evan –create games of each other's heroes and the obstacles they faced. The depicted games are the final versions.

Kenyan program.

I focus on the multi-day “My Partner's Hero's Journey” activity to illustrate more general observations of pairs across both activities. During this activity, a black girl, Jordan, was randomly paired with Evan, a white boy. Jordan appeared to be prior friends with other white and hispanic girls she sat with; in prior activities, she was often patient and engaged and appeared to hold similar knowledge of Scratch to many students in the course. Meanwhile, Evan, who was usually quiet and attentive to helping other students, had corrected the teacher on two occasions, indicating high prior knowledge of Scratch. These students were observed collaborating closely on the first day of the activity, where the teacher remarked positively on their work. On this day, Evan encoded Jordan's hero, her mother and two challenges her mother faced –Jordan's birth at a young age, and attending college –while Jordan encoded Evan's hero, a teacher from the school who had experienced bad teachers (Fig. 3.4). After four days' break in between the continuation of the activity, the researcher interacted several times with the pair:

As I pass by, Jordan asks if I want to see her maze. She clicks the green flag and moves her sprite, a woman, to the end of maze where she reaches another woman –whom she says represents a ‘good teacher.’ But the sprite says nothing. She appears confused. Noticing this state of affairs, Evan starts to explain something to her, then shifts the laptop towards him and shows her how to fix the code.

In this interaction, despite focusing on his own screen, Evan noticed Jordan was confused and sought to help her; he had been observed doing so on the previous day when she was unsure how to color a backdrop. Note that in earlier, standard pair programming with a different white boy, the other boy had dominated control of the computer, even after multiple reminders to share control by the teacher and researcher, leaving Jordan with less time to practice on Scratch. Now her partner shared expertise without removing the ability for her to work independently. But the narrative of the game could be lost during the interaction:

After she debugs the issue, Jordan calls me over to playtest her game again, but I ask her what the story is about before we play. She looks around for Evan’s worksheet; he sees this and hands me his sheet, but I ask her again if she remembers. “I know it’s been awhile.” Jordan says suddenly: “it’s about his teacher’s story going through the sixth grade and her challenges of [encountering] a bad teacher and becoming a good teacher.”

Here, having spent four days away from a single day’s activity, Jordan appears to have forgotten the content of her partner’s maze, despite appearing engaged in finishing it. Under pressure, she remembers the story. Later, Evan

expresses a similar forgetfulness:

I notice that Jordan and Evan aren't interacting as much today –each is focused on their own screens... After walking around, I return and ask Evan to show me his maze. First I ask him what it's about. He looks for Jordan's worksheet on the table top, finds it, and explains using her worksheet as a guide: "Oh, [Jordan's] mom had her at a young age, and then [her mom] went to college which helped her."

There are multiple potential reasons for why students might forget, such as the delay or, possibly in copying Jordan's drawing into the game, Evan overlooked the personal narrative the game represented. Yet partner's hesitation might also be interpreted as *nervousness* around misrepresenting their partner's narrative (e.g., after the class, the teacher stated, "Look at how well they were working. I think they didn't want to screw up each others work"). Indeed, even in the other U.S. class which was typically noisy, students appeared noticeably quieter and engaged during the latter days of the Hero's maze activity. Yet the next interaction with Evan surfaces how preparatory privilege might cut off the potential for deeper bonding or understanding:

As I leave their pair, Evan smiles and tells me that he's been working on something "more complicated than this" outside of class that he's proud of. "It's a platformer game." I ask him what it's about. He says it doesn't have a story like this game.

Shortly after and into the following day, Evan was seen working on that personal game instead, while Jordan continued to expand her partner's maze

by adding a purple obstacle that disappears when the protagonist touches the word “6th grade.” (This behavior mimicked a feature Evan had added, where Jordan’s mother, after touching Jordan’s sprite and the word “college,” would remove yellow obstacles.) Rather than seeking Jordan’s critique of his representation or using it as an opportunity for further collaboration, Evan reverted into individualized play and signaled a belief in what is “more complicated” as having higher value (possibly to signal his competency to an authority figure whose presence represented the white male majority in CS), rather than powerful narratives or social bonding. Though Evan helped his partner and, at least on the surface, learned more about her background, he signaled that he valued code complexity and demonstration of technical competence above all. Later that day and into the next, Jordan was eager to demonstrate her learning to the researcher and teacher; however, the exchange of experience was still limited and reproduced power dynamics (albeit perhaps in less extreme ways).

Although this muted interaction around My Partner’s Hero’s Journey was similar to other pairs, another reading suggests this activity’s design could be improved to address the later behaviors we observed. In the activity, individuals translated their hero narrative into a maze drawing before passing it to their partner, concretizing a copyable blueprint. In an alternate design, their partner could be made responsible for the translation process, requiring regular dialogue between partners. Moreover, students could be asked to present their game in an open playtest –holding them accountable to speaking respectfully and accurately about their partner’s hero and framing the activity as a site to practice understanding and empathy, in addition to computational knowledge of “broadcast” blocks and event handling. Even so, still broader pedagogical changes may be required for making the shift towards relationships in comput-

ing classrooms.

3.4 Overall Thoughts on Fostering Bonding Across Difference in Intro Computing Programs

The story of Jordan and Evan suggests that U.S. students, particularly in dominant social positions, valued the demonstration of technical knowledge above all, signaled by Evan’s comment about knowing “more complicated” coding skills. Rhetoric about tech pipelines, Eurocentric and individualistic notions of success (and detached abstraction) that devalue community and environmental connection [284], as well as media representations of white male “hackers,” are important dynamics Evan may have been reflecting. The U.S. study targeted intergroup bonding with changes to existing activities over Scratch, but intergroup goals were rarely made explicit to participants through deliberate scaffolding and targeted activities. By contrast, the Kenyan program made intercultural learning an explicit goal of the class.

In retrospect, these design choices may be significant. In the Kenyan program post-survey, all students were asked what they had learned in the course before being asked more specific questions about intergroup bonding. 20 out of the 40 students reported learning intercultural competence or communication skills alongside computing knowledge (e.g., *“I learned how to create a game and how to interact with other people from different origins”*). Of the 38 students that specified ethnic origins, 32 reported making at least one cross-national friendship, 26 wrote at least one cross-gender friendship, and only 6 reported solely intra-gender, intra-national friendships. There were indications that these

friendships did not naturally arise by virtue of a diverse classroom. For instance, when asked whether she learned anything from the course important to her everyday life, a Rwandan girl replied: *"Most of the work you do in a group. You have to communicate. And you also have to understand each other. You're taught to appreciate each other's opinion."* The student further explained that most of her new friends were made not from pair or group programming, but from the warm-up activities *"because you had to ask each other questions."* Similarly, when a Kenyan boy was asked why he was making new friends including Congolese, Rwandan, and S. Sudanese students, he stated: *"Because [the teacher] was making us interact with each other. Each and every day he made us into new groups.... Even when he brought up physical games [like Mancala] we got to interact."*

Along with students, Kenyan teachers frequently cited the warm-up, icebreaker activities for contributing to intergroup bonds. For instance, when a teacher was asked whether students would have bonded as well in other, more traditional computing classes, he replied: *"No, no, no. It's not that much [in other classes] because there's no warm-ups. There's no converging aspect [of friendships] because everyone knows... that this is my computer."* In other words, screens distract students from interaction and solidify seating arrangements; class activities apart from the computer that encouraged interaction seemed to counteract these effects. Other teachers reported observing students become more open to interaction as classes went on and attributed this change both to time spent in the class (about five weeks) and the non-computing icebreaker or warm-up activities.

Beyond warm-ups, teachers in Kenya also employed various strategies to encourage students in intercultural learning goals, such as appealing to their own

experience with cultural difference, their shared refugee status, being culturally responsive to the shared local context (e.g., seeing someone get stung by a scorpion for an activity about empathy), or making explicit connections between the conflict resolution activities and class dynamics. For example, one Kakuma teacher deliberately grouped 3 students from the Nuer tribe in the middle of the room, and added a Somali student who knew Swahili and English. Then he asked the group to encode a conversation in Scratch, and told the surrounding classmates to watch. Predictably, the Nuer students spoke in Nuer and the Somali student *“started complaining.”* When a boy tried to translate for her into English, his Nuer partners shifted him back to their tribal language. The teacher turned to the class. *“So, how do you feel if you meet some people, they’re only talking in their languages but you cannot understand?”* This occurred right before an activity where groups of students would encode a conversation about cultural difference between sprites in Scratch. The teacher’s pedagogy is reminiscent of Werner et al.’s strategies for positive pair programming [424].

The relative importance of intercultural competence activities and rhetoric during the Kenyan program, contrasted with the findings of the U.S. study, strongly suggests that *collaborative computing activities, by themselves, will not produce equitable interactions or inclusive climates.* Rather, positive interactions in the Kenyan context appeared to result from the unique *combination* of computing activities with intercultural competence activities and rhetoric. The intercultural focus primed participants to take on the responsibility of bonding across difference and creating inclusive climates. They then practiced this learning themselves during computing activities, especially in moments of breakdown, friction, and gaps in structure.

3.5 Conclusion

Overall, I saw evidence that computing education provides unique opportunities to support intercultural learning. I also found that many moments of intercultural learning were contingent on sites of friction, breakdowns, and gaps, whether planned for or around, or as unanticipated conflicts among the program, participants, and technology. Finally, I encountered a number of instances that complicate any straightforward relationship between computing and intercultural learning. Across studies, teacher capacity was the most important factor in the degree to which intercultural learning takes place. Teacher capacity was often decisive in how well they attended to the program's intercultural goals, managed conflicts or obstacles, and led project-based computing activities. This is consistent with prior literature in technology and education [138, 403].

My findings align with a CHAT perspective on diversity [177, 94] in exploring how computing curricula and structure, coupled with the local realities of implementation “[provided] resistance and affordances to [its] attempts to reach” its intercultural objectives [228, p. 66]. For instance, the computing activity system provided certain *affordances* for intercultural learning and encounter, such as promoting shared laughter through breakdowns, and frictions between pairs negotiating for use of devices. I also found that *resistance* to the innovation's intercultural goals could present critical opportunities for learning in their resolution. Following Engeström, situations involving resistance may be viewed as “contradictions” embodying “the clash between individual actions and the total activity system” [132, p. 98]. In reference to Cole's work on the Fifth Dimension, Gutiérrez *et al.* emphasize how these “contradictions,

experienced by us as conflicts” can be “a major source of change” [177, p. 217] and how diversity may be used “as a resource” to address program goals, *provided participants are primed to see it as a resource*. Said more explicitly, a CHAT perspective recognizes how the many frictions, breakdowns, and gaps present in classes – experienced often as “conflicts” – may instead be mined as resources for intercultural learning, rather than detriments to fully “solve” or overcome. In addition to this straightforward way of looking at intercultural computing programs, I found ambiguous situations which appeared to be affording and resisting intercultural objectives simultaneously, but in different ways. For instance, the actions of participants to bond cross-culturally over out-grouping others, or to practice new behavior prohibited by their culture is, locally, certainly intercultural learning, yet viewed from a broader vantage point may constitute negative outcomes involving reinforcing xenophobia, ageism, or cultural assimilation.

Justice often requires coalitions of people to come together across difference to fight for a common cause. Some of the most important issues –climate change, economic inequality, access to healthcare, and the refugee crisis, to name a few –will require broad coalitions to advance solutions. But to build coalitions strong enough to challenge power requires understanding and respect even across disagreements. Rather than framing all friction as something to be avoided, people must be able to interact even when interactions are initially uncomfortable and risk conflict. It is the resolution of that emergent, intercultural friction which facilitates transformative change [137].

Computing education is still a relatively young field. Understandably, the vast majority of research has focused on designs and strategies for improv-

ing conceptual knowledge and attitudes of all people towards computing, especially for those traditionally under-represented in U.S. contexts. Scholars have developed pedagogy to engage youth, whether by appealing to their interest in social justice, valuing their insight and knowledge, or responding to their cultural practices [365, 294, 127]. These shifts are powerful and important work, and much more needs to be done, particularly with professional development for teachers around culturally-responsive and justice-oriented pedagogy. Where the intercultural approach differs is its chief concern about the quality of social bonds across difference, and how to design computing spaces as positive sites for fostering such bonds. I am still concerned about the quality of CS instruction and justice-oriented activities, but my objective is rather to view computing spaces as a *medium* through which social and culture exchange, learning, and bonding occur between and among students that counteracts divisions present in the wider society. As with Allport's contact hypothesis [328], the question is thus not whether to apply an intercultural approach, but rather *in what context, with what class composition, and with what structure* building bonds across difference becomes viable.

Although U.S. and Kenyan contexts are different in myriad, important ways, perhaps the most salient difference across studies was the preparatory privilege of some white students, which resurfaced old tensions between intergroup contact and unequal dynamics downstream from structural inequities. Like other scholars, I again observed unequal interactions over group activities [271, 294, 366, 416], but I also observed scenarios where students appeared to bond even over unequal dynamics. This reminds us that intercultural and computational learning goals do not automatically align –e.g., there may be scenarios where bonding occurs at the expense of other learning goals –which chal-

lenges us to design solutions which mitigate power differences while continuing to provide opportunities for intergroup contact. My trial of a new activity type, *interdependent programming*, seemed to soften negative power dynamics while still enabling contact, but deeper understanding and empathy across difference appeared muted, and while the activities could be better designed and guided in future work, I suggest the explicit ‘intercultural’ framing of computing classes (and their learning outcomes) appeared to be a significant factor in some of the new friendships that students reported in the Kenyan program. This suggests that future designs should explore framing intergroup interactions as *challenges* for students to embrace and involve activities which teach them *how* to attend to friction that emerges. Computing activities then *become* –and are not automatically by virtue of diversity or justice topics –sites for students to practice attending to emergent friction and build solidarity across difference. Most importantly, students can then utilize their own agency in advancing solidarity across difference –not just during group activities, but in gaps in instruction and outside the classroom context entirely.

CHAPTER 4

DESTABILIZING PROGRAMMING IN CULTURE: “CULTURAL ALGORITHMS” & INTER-CULTURAL DEVELOPMENT

My ethnographic fieldwork of programming classes inspired several lines of inquiry. For one, I became dissatisfied with how little the material aspect of programming –the hardware and the software environment –were conceptualized as cultural and historical artifacts, carrying their own assumptions and values. This question was all the more salient in Kakuma refugee camp, where many East African students had never touched a computer before, let alone programmed one. I eventually wrote a historical piece to deal with the cultural heritage of programming, presented in Chapter 5. However, aside from the material, other questions emerged around the physical activities that required participants to act out code or procedures.

How does one draw a line between what constitutes learning “computational thinking” [430] and what constitutes learning intercultural competence? Although I had only observed two activities where this blurring of distinctions occurred, later activities like “To Market to Market!” –which I was unfortunately never able to observe directly –gave two teams different sets of rules and procedures and then asked them to interact, leading (quickly, and by design) to communication breakdowns of the kind encountered in intercultural communication. In such activities, where did computational thinking end and intercultural learning begin? Are there aspects of programming that might lend themselves well to helping students become interculturally competent? And can this understanding of culture facilitate not just more “explicit” or obvious aspects of difference —like language or dress —but tacit ones, like non-verbal behavior

and social organization?

My inquiry around such questions led to the development of a phrase, “cultural algorithms,” which seeks to unite algorithmic and cultural difference. I later learned that “cultural algorithms” was a term used by Orlando Patterson, drawing from the work of Hutchins. The language he uses evokes computation:

“Procedural knowledge is of two broad types: routines or scripts, and distributed knowledge. Scripts are *cultural algorithms*: stored knowledge of a ‘predetermined, stereotyped sequence of actions that defines a well-known situation’ (Schank & Abelson 1977, p. 41)... Routines can be further subdivided into two broad types: individual and divisional. Individual routines are those performed by a single person such as learning the recipe for food preparation. Divisional procedures, or drills, are those that require alignment with others such as learning one’s role in an army parade, navigation team, or orchestra. [One needs] only know what to do when certain operations occur in the environment and need have no knowledge of the entire script. They simply ‘do X when Y.’ ” [321, p. 11; *emph. added*]

In this section, I propose and explore a new type of activity which mobilizes the concept of “cultural algorithms” to teach about society. The goal of cultural algorithms activities is to help participants externalize and *write down* these procedural scripts in the form of psuedo-code (or actual, runnable code), which can then be reflected on and edited (“changing society”). Unlike ethnocomputing –which connects computing to more “explicit” aspects of culture like language, craftwork or fashion –cultural algorithms activities also include more “tacit” aspects of culture, aspects that have become so normalized, their destabilization

as social constructions can come as a shock to participants embedded in that culture.¹

I present one example of a “cultural algorithms” activity, Birdcraft, that targets a tacit ontology central to U.S. society –the seeming naturalness of “race,” accomplished through what the Fields call “racecraft” [142]. I first outline the “racecraft” perspective on race, comparing and contrasting it with other work on race/ism and HCI. This theory section may itself be a contribution, for, as we shall see, it differs slightly from other prominent perspectives. With a colleague Ariam Mogos, I then outline a preliminary study of activities on cultural algorithms, particularly “Birdcraft” (a play on the term *racecraft*). I draw preliminary findings from a small middle school class over Zoom, and with C.S. education specialists as part of a summer workshop. Connecting with sociologists such as Abena Asare who teach race as a social construction [25], I argue in favor of a constructivist pedagogy which guides participants to come to their own understanding of the social construction of race, rather than a top-down approach of telling, which can provoke denial, defensiveness, or hostility, especially when the person lecturing is implicated in the very social categories they are attempting to reveal as a fabrication. Overall, however, I argue that cultural algorithms activities require some additional setup and follow-up activities to connect them back to the real-world.

¹Unlike the work of Chun in *Programmed Visions* [85] –who might frown upon the notion of ‘algorithm as metaphor’ –cultural algorithms instead embraces that very notion, asking us the question: to what extent was ‘algorithm’ merely a ‘metaphor’ to begin with? When one considers discriminatory practices like blood quantum laws on descent for Indigenous peoples, is the algorithm there really just a ‘metaphor’? To maintain this distinction between algorithm and metaphor –or really, algorithm and culture –we are assuming an underlying distinction between the artificial and the “natural” aspects of culture; we are imagining ‘algorithm’ as a concept detached from history, emerging from nowhere. I think such fears are misplaced, and we might do better by embracing –if tenuously –the concept of algorithm, and repositioning it towards positive ends [2].

4.1 Racecraft and HCI

“If race lives on today, it does not live on because we have inherited it from our forebears of the seventeenth century or the eighteenth or nineteenth, but because we continue to create it today.”

–Barbara J. Fields [142, p. 146]

In this section, I attempt to establish an intercultural perspective on race/ism in HCI by advancing Barbara J. Fields and Karen E. Fields’ concept of *racecraft* into our lexicon. I present “racecraft” both to put a name to some of the troubling tendencies pointed out by other scholars in HCI [189, 53, 230, 58], and as an orienting tool for HCI scholars looking to guard against the trap of reifying race while fighting racism. In defining racecraft, I introduce several terms and phenomena, such as the race-racism evasion, the performativity of social constructionism, and Toni Morrison’s metaphor of the racial house. In doing so, I build upon, and in some ways complicate, dominant U.S. discourses in anti-racism to broaden the conversation on ‘race frameworks’ in HCI [251]. Throughout, I connect the Fields’ work with scholars who adopt a similar perspective that, following Fraser [146, p. 91], I term *deconstructionist anti-racism* (elsewhere anti-race, humanist, ‘post-race,’ or eliminativist anti-racism [152, 350, 135, 273, 18, 268, 335, 106, 378, 206, 298]).

While agreeing with central tenets of CRT emerging from U.S. legal studies, such as racism’s ordinariness [305], racecraft diverges most strongly around concepts of groupness and identity, arguing that the latter term itself serves to naturalize transitions from an (imposed) identification into a (personal) identity [68]. I analogize the debate between CRT and racecraft to the debate between intersectionality and assemblage theory in woman-of-color feminism [293, 343].

Since much prior work focuses on the ‘artifacts have politics’ aspect of racecraft, I also seek to remind readers on how *politics have artifacts*, or how racecraft can act as a cultural worldview and framing device. I provide several examples of the ways racecraft has appeared in HCI work.

4.1.1 What is racecraft?

Racecraft is the Fields’ word for how racism reproduces itself through a seemingly inescapable cultural rationality akin to witchcraft.² For the Fields, “[r]ace is the principal unit and core concept of racism” and *racism creates race*, not the other way around [142]. Historical materialist scholars like the Fields define racism not as prejudice or prejudice-plus-power, but as an *action* and a rationale for that action. This rationale is “the theory and the practice of applying a social, civic, or legal double standard based on [purported] ancestry, and to the ideology surrounding such a double standard” [142, p. 17]. *Race* is “an ideology that constructs populations as groups and sorts them into hierarchies of capacity, civic worth, and [deserving] based on ‘natural’ or essential characteristics attributed to them [and which] help to stabilize a social order by legitimizing

²Fields & Fields are sisters: Dr. Karen E. Fields studies post/colonial Africa and the 20th century American South. Dr. Barbara J. Fields is a Professor of History at Columbia University. Throughout, I use “the Fields” to refer to both authors. By centering racecraft, I intend to explore their argument in detail, not that I am the first cite them. Benjamin wrote a 2014 review [48] and since starting this work, other HCI scholars cite the Fields in passing [439, 300, 272]. Other sociologists and cultural theorists also compare “race” to witchcraft [312, 152, 18]. Another popular theory is Omi & Winant’s ‘racial formation theory’, often cited among HCI scholars as it is often a core textbook in race and ethnicity courses [189, 86, 53]. The Fields neglect to cite Omi & Winant, which careful reading of their work and interviews reveals is not an oversight. Unlike the Fields, Omi & Winant are reluctant to liken race to witchcraft, state race is not a “problem” in-of-itself; that race is “central to everyone’s identity”; and that any attempt to contest race is misguided [309, p. 112]. Others have critiqued formation theory; e.g., Magubane accused Omi & Winant of “American exceptionalism,” contributing to naturalizing and globalizing U.S. racial ontology [267, p. 377-82]; and in Internet studies, Daniels called formation theory ‘theoretically weak’ and warned that it tends to detach race from racism [107, p. 712].

its hierarchies of wealth, power, and privilege” [349]. Without downplaying the “deadly serious problem” of racism, from the racecraft view, “the fundamental problem with racism is that almost everyone –not only ‘the racists’ –acts as if there are people of one or another race in the first place” [207]. The Fields define racecraft not as “refer[ring] to groups or to ideas about groups’ traits” but:

“instead to mental terrain and to pervasive belief... [R]acecraft originates not in nature but in human action and imagination... The action and imagining are collective yet individual, day-to-day yet historical, and consequential even though nested in mundane routine. Do not look for racecraft, therefore, only where it might be said to ‘belong.’ Finally, racecraft is not a euphemistic substitute for racism. It is a kind of fingerprint evidence that racism has been on the scene.”
[142, p. 18]

One of the hallmarks of racecraft is the *race-racism evasion*, where racism “transforms the act of a subject into an attribute of the object.” Race-as-adjective “radiates a semantic and grammatical ambiguity that helps to restore an appearance of symmetry,” often serving to obscure the perpetrators of racism [141, p. 48]. Race is thus “a neutral-sounding word with racism hidden inside” [142, p. 102-3], and “it is the repetition of the act of racism that makes race look like a real entity” [100].

From the racecraft view, algorithms aren’t biased against particular “races” –instead, the bias actually (re)carves the boundaries of races. In a 2014 review, Benjamin rephrased the racecraft perspective as: a black man is not discriminated against because he is black, he is black because he is discriminated against

[48]. Ignatiev & Garvey said similarly: “people were not favored socially because they were white; rather they were defined as ‘white’ because they were favored” [212, p. 1-2]. Thus, from the racecraft perspective, an algorithm does not output a favorable outcome because a person ‘is’ white; rather, a person is (maintained as) white because the algorithm outputs a favorable outcome. The Fields share Browne’s commitment to deconstructing race-thinking by analyzing technology that “reify boundaries along racial lines, thereby reifying race” [67, p. 8].

Racecraft thus provides a useful term between race and racism that calls attention to how (a culturally-specific) racism “assembles” or conjures races [156, 298], forming an “invisible ontology” that transforms “relations between persons [to] relations between races,” leading to seemingly innocuous phrases like “race relations” and “inter-racial” [142, p. 214-5]. The term ‘racecraft’ draws a comparison to witchcraft not to deny the rationality of either, but to “assume it”: “*[Both] are imagined, acted upon, and re-imagined... the action and imagining inextricably intertwined. The outcome is a belief that ‘presents itself to the mind and imagination as a vivid truth’... Witchcraft has no moving parts of its own, and needs none. It acquires perfectly adequate moving parts when a person acts upon the reality of the imagined thing.*” The Fields provide examples of how U.S. onlookers perceive several situations, such as conflicts among college roommates, as “racial,” rather than everyday differences of personality or opinion. While an outsider to the U.S. might attribute these conflicts “to ordinary cause and effect... the American is the insider on the alert for witchcraft” [142, p. 19-22,30,40].

This conjuring trick recalls Joerges’ rebuttal to Winner’s piece “Do Artifacts Have Politics?”, the latter often cited by race and HCI scholars [188, 49, 374,

305, 102]. Joerges asked “Do Politics Have Artifacts?” to highlight that how artifacts are ‘read’ can produce evidence to justify one manner of reading – certain intent is assumed and then reproduced, ahistorically and drained of additional context, as *the* conclusion about the artifact [220]. In such stories, we are “forewarned that things are other than they seem,” motives are “revealed,” and then we are told “*the* outcome” [435, p. 34]. The Fields describe a commotion over a journal cover depicting a (Southern) Black woman, meant to refute the Aunt Jemima stereotype. Readers (in the Northeast) rushed to declare the cover “racist,” though the Fields questioned this: “the portrait [cannot] make dark skin, in and of itself, hard to look upon. The viewer must bring that reaction to the picture or, for that matter, to a real person” [142, p. 72]. So too is racial disparity discourse often “laced with generic, *a priori* assumptions about the role of [race]” where disparities are framed as solely “racial” when they are “in fact embedded in multiple social relations” [351, p. 151]. Such racecraft scholarship “first strews race and races everywhere and then, *mirabile dictu*, discovers them everywhere” [141, p. 51].³

The difficulty is that many technologies and artifacts *do* embed racism and *are* produced from racist structures [301, 49, 67, 69]. Going too far on ambivalence leans towards feeding “uncertainty” [401, p. 15], what Piper calls “impatient musings on the subjective incompatibility of different worldviews” [334, p. 395]. Still others may argue that overly foregrounding the social construction of race may be anathema to the *political* choice of a necessary *strategic* essential-

³For example, in Cave & Dihal, they show how the imagery of AI systems are often snow-white or androids with pale skin. To account for this, they ‘forewarn’ the American reader that “even narratives of an AI uprising that are clearly modelled on stories of slave rebellions depict the rebelling AIs as White” [78, p. 697]. Because the American reader automatically associates ‘slave’ with dark skin, ‘things are other than they seem.’ In fact, the origin of the android uprising is from the 1920s Czech play *R.U.R.* by Čapek. The Czech term ‘robot’ referred to the forced labor of ‘white’ serfs in Eastern Europe.

ism [380] that reifies and essentializes social categories to fight against the social hierarchies that (re)inscribe them [103]. Yet in considering the Fields' perspective, I want to focus on how racecraft acts as a culturally-specific ontology –a particular 'racial' worldview –not because other foci are less important, but because this aspect seems to have received less attention, and becomes especially important as U.S. racecraft and anti-racism 'travels' through media technology. My inversions of the phrase 'artifacts have politics' to 'politics have artifacts' shall be rather broad. To combat the spread of this polarizing 'racial' worldview, the Fields argue to shift towards language of ethnic groups and ancestry, stressing that "the equation of Afro-Americans' peoplehood with race is a corollary of racism" [141, p. 50]. Aligning with this position, in 1997 Patterson concluded that 'race' in the common 'race/ethnicity' distinction is "a distinctively American belief, an essential part of American racist ideology... [it maintains] the binary concept of 'race' that prevails in America... This is its only linguistic function" [320, p. 76]. 'Race,' and its counterpart racial identity, in the U.S. acts as a cultural code to submerge and hide one-drop rules, naturalizing anti-Blackness and anti-Indigeneity by confusing insiders to equate 'race' with phenotype and groupness. In short order, I will provide concrete implications for HCI; first, however, we must grapple with how this differs from other work in HCI and beyond.

4.1.2 Why do we need another race framework?

In 2020, Ogbonnaya-Ogburu et al. published "Critical Race Theory for HCI" at the annual CHI conference, contributing to the discourse on race and racism and/in HCI [305]. Around that time, many other scholars have contributed to

the discourse in HCI, including Ruha Benjamin and Yolanda Rankin [49, 344]. These papers and discourses draw largely on traditions in Critical Race Theory (CRT), a line of inquiry founded by U.S.-based critical legal scholars Derrick Bell and Kimberlé Crenshaw, for understanding race/ism and intersecting discriminatory systems.⁴

Although CRT has been widely influential, it bears mentioning that many prominent scholars who study race/ism –such as the Fields, but also Paul Gilroy, Orlando Patterson, and Adolph Reed Jr., among others –do not call themselves critical race theorists and at times actively distance themselves from the discourse (e.g., [108, 95, 141, 153, 349, 320]). Black feminists such as Emma Dabiri and Jennifer C. Nash also distance themselves from earlier conceptualizations of race and anti-racism in Black feminism, finding more commonality with the identity-deconstructionist work of Jasbir Puar on assemblage theory [106, 343, 293, 292]. Despite this complexity and debate, I have noticed HCI scholars –new to scholarship on race/ism, and reading CRT work for the first time –understandably assume such scholarship is coherent and homogeneous, thereby ignoring divergent perspectives in the philosophy of race, historical materialism, postcolonial theory, cultural studies, or feminism [159, 342]. In a 2020 conversation, prominent race theorists Paul Gilroy and David Theo Goldberg also expressed concern, remarking that the discourse around race suffers from a kind of “presentism” where past debates are ignored and racial ontology is de-historicized [157]. Although the differences between CRT and other approaches vary, one common claim made by detractors to CRT is that some work done

⁴In this chapter, I delimit CRT largely to its emergence from critical legal studies [95, 108, 251] and the key tenets presented by Ogbonnaya-Ogburu et al. [305]. Many scholars herein do not identify themselves as CRTs and sometimes deploy strong critiques; see [108, 95, 141, 153, 349, 320]. Other scholars may not delimit ‘critical race’ studies in this way; however, the term risks both conceptual inflation and creating a false sense of agreement and equivalency.

under its banner suffers from circular reasoning. This concern is related to the concept of racecraft just defined. To illustrate how, it is instructive to recall some debates around defining racism.

Two major types of definitions of racism are prejudice and prejudice-plus-power [205]. A 'prejudice' definition of racism focuses on intergroup conflict between people of different races.⁵ A prejudice-plus-power definition emphasizes that power dynamics are uneven and coded into institutions and culture. Here, racism may be positioned as the defense of racial privilege of the dominant white group, and people of color cannot be racist [205, 119]. Finally, I reflect on definitions provided by Kendi, since his work has received widespread attention, teaches a specific way to be anti-racist, and for some in American popular culture, has become synonymous with critical race theory [233]. Kendi defines racism as "a marriage of racist policies and racist ideas that produces and normalizes racial inequities." For Kendi, "a racist idea is any idea that suggests one racial group is inferior or superior to another racial group in any way" and anti-racist ideas are those which declare "racial groups are equals in all their apparent differences" [233]. A similar definition of racism in terms of racial groups appears in Ogbonnaya-Ogburu et al. [305].

These definitions of racism are extremely valuable, but they have a "limit," according to Tuck & Gorlewski and Patel & Price: they fail to embed how racism produces race [405, 319]. Said Mason in a PhD dissertation at Howard University: "Kendi's inability and articulated unwillingness to consider that race is racism is indicative of the overarching problem related to such discourse" [273, p. 7]. To put the problem explicitly to the reader: racial groups can never be

⁵E.g., Google Search in 2020 defined racism as: "Prejudice, discrimination, or antagonism directed against a person or people on the basis of their membership of a particular racial or ethnic group, typically one that is a minority or marginalized."

equal, because asymmetric cultural rules define them [377]. Definitions that describe racism in terms of discrimination against “racial groups” put the cart before the horse: they embed an assumption that race pre-exists racism, *even while authors may declare that race is socially constructed*. In their circularity, definitions that use race to define racism exhibit racecraft.

4.1.3 The performativity of “race is socially constructed”

“‘Race relations’... as an analysis of society takes for granted that race is a valid empirical datum and thereby shifts attention from the actions that constitute racism... to the traits that constitute race... For scholars in our own time who accept race, once ritually purified by the incantation ‘socially constructed’... the relevant traits are more likely to be ‘difference,’ ‘Other-ness,’ ‘culture,’ or ‘identity.’”

–Barbara J. Fields [142, p. 151]

How could it be that race scholars who otherwise declare that “race is socially constructed” fall into the trap of racecraft? Reciting the social construction thesis does nothing to cleanse authors from racecraft precisely *because* racecraft is like witchcraft: an invisible ontology rather than simply a set of facts [142, p. 220]. Many sociologists have criticized how the declaration “race is socially constructed” can become performative, whether devoid of practical meaning or seemingly not understood by those who declare it [135, 152, 160, 268, 350, 348, 320, 377, 251, 20, 427, 206, 105, 434, 335, 267]. Gilroy calls performative overtures to social construction “the pious ritual in which we always agree that ‘race’ is invented but are then required to defer to its embeddedness in the world and to accept that the demand for justice nevertheless

requires us to enter the political arenas that it helps to mark out" [152, p. 52]. According to Spencer, the obligatory reference to social construction before race-talk may even become a *defense* for racecraft, an absolution ritual followed by a hidden 'carry on as you were' [377, p. 250].

The performativity of "race is socially constructed," what Gilroy calls the move to "rescue race from racism" [154, p. 18], appears in major works in HCI. In her essay *Race and/as Technology*, Chun "frame[d] the discussion around ethics rather than around ontology, on modes of recognition and relation, rather than on being." After pointing out that race is socially constructed, she states that racism "stems from race" and that race "organizes social relationships" – phrases that are signatures of racecraft [86, p. 9-14].⁶ Goldberg reminds us: "Racisms establish, set in place, and extend races, not the reverse" [161, p. 171]. For the Fields, race stems from racism; it is not an independent causal force that "organizes" reality, nor can it be sanitized or liberated [141]. In Chun's closing argument, she claims that "the best way to fight racism might not be to deny the existence of race but to make race do different things." This repeats earlier work by Haslanger ("we might instead ask 'race' to do different things" [194, p. 52]), who also 'moves on' from racecraft views like Appiah's [19]. Where *race* as technology "operate[s] on understandings of how it works that are already in place" [100], racecraft refers to those understandings themselves (mental terrain/belief) and the everyday practices (acting on that terrain) that give race coherence. Racecraft is what Coleman's "hammer" of race hits [96, p. 178-9]. Like

⁶Chun calls the Fields' position "race as culture," citing a book by Reardon to 'debunk' it. In the cited pages, Reardon claims that race isn't "treated as a historical object" by the "critical race theorists," referring to work by Barbara Fields [347, p. 18]. This claim is *factually incorrect*: Fields is a *historian*, and Fields has never called herself a critical race theorist. To be clear, "ideology" for Fields means race is performed, cultural, and rational, not that it is merely a "discourse" or language as Chun suggests from Reardon's misreading [142, p. 137-9]. See Patterson on 'culture' [321].

eliminativists, the Fields wish to target racecraft in addition to racism, rather than only subverting the existing terrain, important endeavors that nonetheless may leave the underlying infrastructure intact [197]. Said differently, the Fields align with Collins' matrix of domination [97], but seek to remind us that, strictly speaking, "race and class do not intersect but racism and capitalism do" [273, p. 242].

This view –on deconstructing race while fighting racism –I shall call *deconstructionist anti-racism*. Where deconstructionist anti-racists differ from others is their *social* and personal refusal, or attempts to deconstruct, racial categories and notions of identity around them. They remark that challenging race's social construction, or transcending what Mahiri calls the "color-bind" [268], can summon perceptions of color-evasion⁷ or non-racialism, policies and rhetoric which has managed to uphold racism, rather than dismantle it [161, 135, 377]. They may also be accused of hopelessly utopian thinking, lambasted as humanists, or exhibiting privilege [338, 119, 153, 422, 6, 287]. Thus, deconstructionist anti-racism is "liable to make one an intellectual punching bag of critics from left to right" [251, p. 153].

The debate between identitarian and deconstructionist anti-racisms has a long history.⁸ While anti-racism "is usually thought of as undoing or revers-

⁷Color-evasion [16] is the claim that one "doesn't notice race"; or if we just stop talking about race, racism will go away.

⁸For those wishing for more precise definitions, I provide them here. Goldberg defined several kinds of anti-racism: *anti-racialism* that "seeks to end racial reference" without attending to racism; *racial anti-racisms* where one fights within racial identities and which sometimes requires reifying race; and *nonracial anti-racisms* that seek to fight racism without racial identity, but "have tended to stress individual rights over group rights... [and] to elevate concerns about racial definition." Goldberg believes that "there is no inherent necessity to the devolution from racial to racist reference," yet later he is more positive about non-racial anti-racism, calling it an important "dream" that critically reorients anti-racism and combats those who express "dogmatic religious profession [that] insist that their [racial] commitments are the best or only worthy way of being in the world" [161, 162-72]. Here, I define *identitarian anti-racism* as racial anti-racism that, though it expresses conservatism, in practice risks slipping into essentialism

ing the political economy of racial sovereignty and superiority... less often there is a call to derail the social ontology and architecture of racial classification... the building blocks of racisms' constitution" [161, p. 166]. Gilroy has long argued for a deconstructionist, humanist anti-racism [156, 154, 153]. Gilroy contrasts his type of anti-racism to the pessimism of many critical race theorists, most notably Derrick Bell, who called racism "permanent" and "indestructible" [395].⁹ One of Bell's colleagues, Kimberlé Crenshaw, argued that embracing racial identity was "the most critical resistance strategy for disempowered groups," chastising humanistic or deconstructionist strategies as "vulgar" and dismissive. She argued that "I am Black" is a positive identity, while "I am a person who happens to be Black" is "straining for a certain universality... a concomitant dismissal of the imposed category... as contingent, circumstantial, nondeterminant." She tempered the strategy as dependent on time and context [103, p. 1297]. Throughout *Racecraft*, **the Fields view the concept of racial identity as an *obstacle* to fighting racism, rather than a cure.** In an interview with Jacobin Magazine, they reveal that they prefer Crenshaw's second phrase and allude to other scholars as "primordialists": "They believe that they are what racecraft made them. Race is an identification... It's not an identity" [100].

By now, it should be clear that critical race theorists and deconstructionist anti-racists differ on some key points around groupness, identity, and anti-racist action. These debates are analogous to debates in postcolonial and feminist

[152, 135, 322, 298]. Deconstructionist anti-racism is anti-race anti-racism, a 'nonracial' anti-racism with a commitment to both fighting racism and deconstructing race simultaneously, including racial identities, that nevertheless risks focusing too little on racism [106, 135, 273]. Most advocates argue 'white' identity must be deconstructed first [211, 106, 422, 377].

⁹To be less subtle: in a passage, Gilroy alludes to CRT as a "U.S.-centric" framework that overcorrects for color-evasive liberalism: "This orientation answers the liberal culture of denial by saying that 'race' is not nothing but everything: a permanent and apparently inescapable feature of society." It is fatalistic and resigned, its proselytizers "extremely attached" to reifying race, appearing "more concerned with arguing that any aspiration to live outside of racialized bonds, codes, and structures of feeling is naïve, misplaced, foolish, or devious" [153, p. 145].

studies on the political value of strategic essentialism over deconstructionist approaches to fighting oppression [380, 293]. Before I close this section, however, I want to reflect on the fact that Toni Morrison advanced a similar anti-racist perspective to racecraft and faced likeminded criticisms. Her concept of the “racial house” will prove useful to framing future discussion and help me reveal why the racecraft perspective may also be considered an “intercultural” approach to racial literacy.

Throughout her career, Morrison was interested in deconstructing race, writing works such as *Recitatif*, *Paradise*, and *A Mercy* that purposely eschewed racial categories. This “non-colorist literature” was not “literary whitewashing,” she chided critics, but a move “to defang cheap racism, annihilate and discredit the routine, easy, available color fetish, which is reminiscent of slavery itself” [288]. In her 1997 essay *Home*, Morrison introduced the metaphor of a “racial house” to describe the paradox of combating racism from within racecraft. In her writing, she tried to make the house more hospitable, but then asks: “Could I redecorate, redesign, even reconceive the racial house without forfeiting a home of my own?” [287]. At the end of *Beloved*, she had written a word that signed a stereotype, yet many Afro-Americans would also use it: the word had the wrong politics, but in changing it, she genuflected to a particular ‘politics’ of the word—the White gaze’s. Race became a self-filling prophecy [281]. She challenged others:

“We need to think about what it means and what it takes to live in a redesigned racial house and evasively and erroneously—call it diversity or multiculturalism as a way of calling it home. We need to think about how invested some of the best theoretical work may be

in clinging to the house's redesign as simulacrum." [287]

Morrison's remarks reflect the Fields' suggestion that the visual salience of U.S. racism presents such a "vivid truth" that even anti-racists and race scholars can slip into the twilight of racecraft. In *Blinded by Sight*, Obasogie argues that people who are blind are better at understanding racecraft, because they notice how race *becomes* rather than simply *is* visible [303, p. 180]. When people visit who are far outside U.S. culture, they also notice this becoming [397]. For this reason, I have come to believe that operating from a "racecraft" perspective requires an intercultural understanding of race/ism in the U.S. Those who are born and grew up in the U.S., embedded in the society, are unlikely to attain this level of intercultural competence from experience in that society alone. History bears this out, showing many examples of famous Black Americans who broadened views on race after journeys abroad, including Malcolm X and James Baldwin [268, 39, 437]. Many scholars who speak of deconstruction also draw from intercultural experiences [152, 427, 206, 335]: Piper stated that visiting Germany "taught me what it meant to be an American" [337] and the Fields themselves claimed that to understand racecraft, "the necessary first step was that we had to have some kind of intellectual detachment from American society" [100].

If understanding racecraft requires one to draw upon intercultural experiences to "get outside" the racial house, it is understandably the case that acting on such a view from *within the society it derives* can incur suspicion or outright hostility [179, p. 59]. Asare, a sociologist who teaches race, echoes this point, suggesting that the denaturalizing of race "carries with it psychic risk" for Black students, that there are "costs" to attempts to live outside it, and likened its re-

jection to an “exorcism” that would “[cast] out racial essentialism” [25, p. 22-3]. Intercultural theorists have repeatedly noted that when pointing attention to the means with which people make sense of their world –from which they derive community and meaning –people are liable to react with defense, hostility, or confusion [52, 183, 334]. Adrian Piper’s concept of *literal self-preservation* argues that we are predisposed to preserve our (racial) rationality by shuffling away threatening *anomalous data*: “compelled either to conceptualize the objects of our experience in familiar terms, or else not to register them at all... this is a necessary condition of preserving the unity and internal coherence of the self” [334, p. 447]. Only when normalized rules are broken may we realize they exist [183, p. 93]. And so, when presenting views similar to racecraft, scholars have worried and seen that it may be particularly charged for Black Americans [25, 155, 206, 379, 160, 273, 320]. Upon reading *Racecraft*, Ta Nehisi-Coates stated that it was “a challenge to African-Americans who have accepted the fact of race and define themselves by the concept of race”;¹⁰ Gilroy was confronted by Black American parents who demanded that he be “stopped from teaching [their] children” [158]; Hoyt begins a presentation with a slew of disclaimers [207]; and Spencer claims that he has been ignored by many of his contemporaries [379]. Even recently, a talk by Adolph Reed Jr. was contested by a young AfroSocialist group [339]; as reasoning, the group stated that “race isn’t bad in and of itself. Racism is bad and needs to be destroyed” [4].

¹⁰Some passages from Coates’ *Between the World and Me* are eerily similar to *Racecraft*, which he was reading at the time of writing. While enjoying his writing, the Fields later perceived that Coates failed to internalize their premise [100].

4.1.4 Past work in HCI, critiqued through the lens of racecraft

“I messed around with children, words, water, Coke, peanuts, bottles, bowls, beam balances, tape recorders, translators, transcriptions... and so on, ordering them all into an almost smooth operation. Nothing would have happened without my energy, my organizing, my bringing and carrying, my telling others to do this and to do that, my arranging by putting this here and that there, saying this and that with a zealous and obsessive bossiness.”

–Helen Verran [412, p. 146]

In HCI, the framing of race as a levered mechanism or reclaimed technology has proved productive and inspiring [96, 86, 49]. Yet to use race as a lever depends on an existing cultural terrain which renders that understanding of “race” coherent. For advocates of the racecraft perspective, the doing of race *is* the what of race, the how of race *is* the knowing of race –they are not separate as Chun suggests [86, p. 8]. This debate is reminiscent of a similar debate between intersectionality theory and assemblage theory in woman of color (WOC) feminism. As chronicled by Nash, the debate stems from how advocates of intersectionality theory tend to make separations between “who people are” and “the way things work” (identity and practice). She recounts that such a dichotomy is false –that “who people are” can never be understood apart from “the way things work” –and that such a distinction may derive from inexperience with how other societies work [293, p. 75]. From the racecraft perspective, race cannot “act” outside of the cultural framework that renders it coherent. In other words, Coleman’s example of President Obama’s speech [96], where Obama is viewed by insiders as fluidly shifting between racial ‘codes,’ is totally ‘en-

crypted' to someone outside U.S. racecraft [265]. While within the racial house, race can function as a *trapdoor*, the Fields view race ultimately as a *trap*.

I now explore three examples of how racecraft operates in HCI: race as phenotype, race as culture, and race as identity. What characterizes the racecraft approach is centering how humans and technology *produce* race, repositioning race as an *outcome* of an interaction, rather than a cause. Often, this inverts commonsense cause and effect. When racecraft acts as a worldview of the author(s), it is not necessarily the artifacts under study that “have politics,” but rather, that the authors’ politics “have artifacts.” Since others provide more historical accounts of race-thinking [53, 189, 58, 69], I use concrete examples, not to single out authors, but as a pedagogical opportunity to explain the perspective. Note that the underlying phenomena are *widespread* across HCI –and not only HCI.¹¹

Examples of racecraft in HCI

The most common racecraft is the conflation of race and phenotype. Despite repeated warnings [53, 69, 236, 66], it continues to plague HCI studies and datasets. We find that AirBnB host race “influences” the price of AirBnB listings [218, p. 9]; the “race” of hands “impacts” purchases on eBay [29]; or “the person’s race influences” how angry they are perceived by emotion detection algorithms [355, p. 1,6]. Applying racecraft, ‘black’ people are not more likely to be perceived as angry; people are ‘black’ because the algorithm perceives them as angrier, relative to other faces. While categories can support quick analysis, I urge researchers to consider reframing such studies in terms of *producing* racial

¹¹For instance, analyzing 63 public documents from major tech companies, Hamilton found that they were saturated with racecraft [185]. Note too that this represents the *racecraft* perspective, and not all views; for instance, Benjamin [49] seems to conserve aspects of racial groupness (understandably, perhaps because she is writing to a popular American audience).

difference [447, p. 95]. “Neither ‘witch’ nor ‘pure race’ has a material existence... Both are products of thought, and of language. Having no material existence, they cannot have material causation” [142, p. 22].¹²

More pernicious is the conflation of race and culture [142, p. 102]. In 2020, computer scientists at Stanford and Georgetown published a study in PNAS, a prestigious journal [242]. Comparing datasets of small samples of “white speakers” from California and “black speakers” from the eastern U.S., they plotted race as independent variables and found ‘racial disparities’ in recognition of Afro-American Vernacular English (AAVE). The location with the highest word-error-rate for ‘black’ speakers was Princeville, NC, named after Turner Prince, a freed slave who, after the Civil War, built houses for other freed slaves near Union stations.¹³ Another location, Rochester NY, produced no significant difference from ‘white’ speech, which they call an “anomalous” result [242, p. 7685].

At first glance, U.S. readers may wonder what, exactly, is wrong with this study. After all, a “racial disparity” was found between “white speakers” and “black speakers” –isn’t this a socially beneficial finding? Said Reed & Chowk-wany:n:

“Our overall concern is the extent to which particular inequalities that appear statistically as ‘racial’ disparities are in fact embedded in multiple social relations and how the dominant modes of approach-

¹²When positioning race as a cause, I often see HCI scholars cite books that say doing otherwise is “colorblind” racism (e.g., Kleinberg et al. & Yu et al. [240, 440]). This confusion between racism and race seems to come from how well-meaning race-conscious approaches may unwittingly naturalize race.

¹³Princeville is the oldest town incorporated by Afro-Americans in the U.S., vibrant and thriving since 1865 in spite of pervasive Southern racism. On the webpage to the town, residents lament how when researchers visit, they seem totally unaware of the town’s history [402].

ing this topic impede the understanding of this larger picture. We believe that too much writing... is laced with generic, *a priori* assumptions about the role of racial categorization that then straitjackets research and tempts researchers... to 'load the dice in favor of one type of description', in this case, characterizing disparities in outcome as strictly 'racial' and thus resulting in the ho-hum and one-dimensional research conclusions we have mentioned." [351]

Consider how the racial framing obscures, then, the fact that the "white speech" sample is assumed to be representative of the entire "white" population of the U.S. –over 200 million people –even though only a small segment of white Californians are represented. This small segment from the West Coast is then compared to similar small segments of black participants from the *North-east and Southeast U.S.* The part of the sample from Rochester, NY shows no disparities with white Californian speech, so the authors dismiss it as "anomalous" –i.e., it is surprising to the authors that the speech recognition system had no trouble with recognizing black New Yorkers, when they are (assumed to) speak very differently than white Californians. Any other 'races' of speakers –the 'anomalous data' –the authors shuffled away by excluding it *a priori* from their comparison, exactly as such data needs to be shuffled away for people to maintain their 'racial' rationality [334, p. 447].

In other words, it is entirely possible that a significant part of the disparity was due to the difference between West Coast and Southern accents –this possibility is never accounted for, however, even in the supplementary materials, because it might have never occurred to the authors or the reviewers to account for it. Instead, we are told '*the outcome*' of the speech recognition technology is

'racial' disparity, simply by virtue of Afro-American involvement. "Segregation disappears as the doing of segregationists, and then, in a puff of smoke –*paff* –reappears as a trait of only one part of the segregated whole" [142, p. 17]. The actual outcome of many studies is to *produce* race, "constantly churning out factitious evidence for an ever-expanding American immensity, the so-called racial divide" [142, p. 24]. Had the comparison group to the 'white' Californians been 'black' Nigerians, one wonders whether the disparity would be called 'racial' –or, for that matter, 'white' Appalachians, Newfoundlanders, or Louisiana Cajuns. Not only do datasets have politics, politics have datasets.

Even sneakier is the widespread U.S.-centric assumption that everyone 'has' a racial identity. The Fields are steadfast in shifting towards language of local ethnicity, not race, to separate confluences of identifications and identities [68, 264].¹⁴ When people talk of race in HCI, they often adopt racecraft language, e.g. "No matter where you live, race makes an impact on your life," "there is no outside of race," race is "one aspect of identity" [374]. The Fields remark that "the very phrase *accurate racial identity* ought to set off sirens. Dangerous lies do not always dress the part" [142, p. 3].

Across HCI, intersectionality theory has deployed the concept of racial identity, often setting it on-par with other identities, such as gender or class [373, 305]. Intersectionality theory embodies long traditions in Black feminism and is a critically important tool and *starting point* for analyzing structures of oppression [99, 293, 344]. Yet we find scholars, even accomplished ones like Omi &

¹⁴The Fields' 'identification' differs from Stuart Hall's, for whom it is a personal "recognition of some common origin or shared characteristics with a person or group." Hall's concept of identity –which he admits is "not widely shared" –appears more like the Fields' identification, an imposed category [184]. Unfortunately for Hall, in common discourse today, *identity* is not meant as an ascription, but rather a personal identity, often by placing it alongside gender, religion, etc. It thereby ensures the racecraft that transforms a description into a classification escapes notice.

Winant and Collins, claim that race is “central to everyone’s identity,” “constitutes a fundamental aspect of human identity” [309, p. 112,246] or “[e]veryone in this room has a race/class/gender specific identity” [98, p. 28]. From the Fields’ perspective, “race as identity breaks down on the irreducible fact that any sense of self... is subject to peremptory nullification by forcible extrinsic identification” [142, p. 157]. Thus “a Puerto Rican, used to not being classed as black in Puerto Rico... may find herself suddenly identified as a black person” in the U.S. and discriminated against [418]. Said Puar in a somewhat (in)famous critique [293]:

“Intersectionality demands the knowing, naming, and thus stabilizing of identity across space and time... [it] den[ies] the fictive and performative aspects of identification... [and it] colludes with the disciplinary apparatus of the state... in that “difference” is encased within a structural container that simply wishes the messiness of identity into a formulaic grid... Identity is... a capture that proposes what one is by masking its retrospective ordering and thus its ontogenetic dimension –what one was –through the guise of an illusory futurity: what one is and will continue to be...” [343, p. 212-6]

Intercultural perspectives are important here. When a BBC reporter went to interview members of the so-called “Coloured community” in South Africa, one man challenged the premise of her visit: “History has been distorted, so much so that even the Coloured believes he’s a Coloured. Even though this identity was forced upon us by the colonialists” [75].¹⁵ In another example, a

¹⁵In the U.S, an anti-racist activist, Starlette Thomas, eerily echoes the words of that man: “We’ve been told that we are colored people [sic] for so long that we cannot even imagine who we would be apart from race. It is impossible to consider because ‘this is the world that we live in.’ We don’t believe that we can change it so we allow it to change us” [399].

'black' Japanese woman claims she "saw color since birth, [but] never saw race. It was never an issue. And I learned it going to America" –despite growing up surrounded by kids who were phenotypically 'Japanese' [397].¹⁶ Her Afro-American parents explain:

"When we took [our kids] to church [in Japan], and met new people, and we asked who they talked to, they would say... 'Well, she's pink. Or light pink. Or... dark brown'... And so when we went to America one day, they came home and said something about somebody being black. My heart dropped. 'Oh here it goes. They learned the white/black thing.' And I said to my son, you know... I've never met anyone black." [398, 12:00]

How did the woman's 'race' intersect with gender prior to the U.S.? No one can answer without equating a description with a classification. The woman goes on that the "hardest part" of being in America was that "people are always going to ask me questions and I have to be ready with an answer with an identity" [397]. Americans exert intense pressure: they somehow *need to 'know'* one's race. In their 'need to know,' they *create* race.¹⁷ Historically, 'identity' emerged

¹⁶Living in a rural village in the 90s, surrounded by a small community of 'Japanese' kids, she claims, multiple times, she was never discriminated against. Note that in many other places in Japan, there *is* discrimination against 'foreigners,' which can intersect with 'black' racialization [24, p. 63-5]. The *hafu* ハーフ seem the most discriminated (although this is changing), but it is not limited to anti-Blackness [362, 361]; for instance when Nippon Airways decided to don "whiteface" and Ariana Miyamoto's best friend, a half-'white' *hafu*, committed suicide from school ostracization [431]. This general racism towards 'foreigners' may appear alien to U.S. observers. Imperial Japan's concept of *minzoku* 民族 appropriated Western scientific racism to justify a caste hierarchy imposed on Burakumin, Korean, Chinese people, etc. [231, p. 27-8].

¹⁷Adrian Piper states that announcements of 'racial' identity "put everyone in their place" by "attempt[ing] to racialize their audiences... as either conforming to or diverging from that 'racial' identity. These attempts fail systematically and by definition... Rather, [they] reif[y] those crude racial stereotypes into an unconvincing simulacrum of social reality in an obsessive-compulsive ritual of wishful thinking" [336].

from psychoanalysis to mean *sameness* and rose in the 1960s United States for politics and consumerism [283]. Race-as-identity, mistakenly serving as a synonym for phenotype, once more hides how race *becomes* [303], reinstates U.S. dominance, and naturalizes the products of racism. Worse, the concept creates a recurring issue for those who have rejected it: for instance, when HCI scholars call Indigenous a “race” [373] or “Native American” a “culture” [428], conspiring with blood quantum policies to re-instate Native peoples as a racial group [23, 329, 363, 126, 405, 244]. One Apache woman remarked of blood quantum: “my sisters are short 1/16 of a degree. What does that *mean*? Does that mean their pinkies aren’t Apache?” [383].

Widespread across the U.S. and HCI education, works in anti-racism such as Tatum [394] forward theories of racial identity that, making U.S.-centric assumptions, tend to presume such an identity will or should arise. Hoyt introduced a framework of “non-racial” identity and argued that racial identity must consider agency. Mobilizing a metaphor of the “room of racialization” that echos Morrison’s racial house, he acknowledges the potential empowerment of racial identity, but wishes to guard against pathologizing its embrace [206]. Throughout education, Tatum and others’ theories of race-as-identity hold sway, and complications like Hoyt, Spencer, or Puar’s are given little to no attention, dismissed, or suspect [293, 377, 343, 206], even regarded as traitorous [342, p. 53]. In the process, educators who ascribe race to children overlook their part in the perpetuation of racecraft. The Fields remind us that in trying to define race, Du Bois “repudiated all efforts to define race as a characteristic or attribute of its victims... whether the definition hinged on biology, culture, or identity... The black man is not someone of a specified ancestry or culture... A black man ‘is a person who must ride ‘Jim Crow’ in Georgia’” [142, p. 158].

4.2 Birdcraft: a cultural algorithm activity for racial literacy

Although I could certainly continue applying the racecraft perspective to other work in HCI, I would like to shift gears now back to our original topic of concern: using the concept of algorithms as a mechanism to reveal cultural ontologies like racecraft. Like Benjamin's focus on racism as an "input" to tech [49], rather than asking how culture is embedded in algorithms, "cultural algorithms" asks how algorithms are embedded in culture. 'Cultural algorithms' does not mean that humans are rigid automatons; instead, this perspective adopts Abebe et al.'s algorithm as formalizer [2] to position computational practices and tools as an integral part of analyzing the construction of societies. The idea of using cultural algorithms to help participants deconstruct social categories was influenced by Bekerman's work in critical peace education, which argues that peace cannot be imagined from within certain societies' social constructions [47].

4.2.1 The Design of Birdcraft

To help students get outside their heads, we designed a fictional bird world. Island A has birds with relatively short legs, Island B has birds with longer legs. One day, birds on Island A got greedy and flew to B to get more worms. They imposed a caste system on the Island B birds, defining a sorting procedure for grouping who is Short and Tall, a descent algorithm to sort offspring, and the attributes and treatment of each "group." The worksheet for Birdcraft is depicted in the full-page Figure 4.1.

|> Birdcraft

Your names: _____
 Your bird's caste: _____ inches
 Your bird leg sizes: _____ inches

1 Select some attributes to categorize on



2 Sort into group based on attribute ("leg size")

```
PROCEDURE SORT_BIRD(bird):
  IF bird's leg > 3 inches THEN:
    Return TALL
  ELSE:
    Return SHORT
```



3 Assign traits to each group

```
IF bird is TALL THEN:
  Weird, Scary, Thief, ...
IF bird is SHORT THEN:
  Beautiful, Smart, Leader, ...
```

4 Essentialize differences: make caste hereditary through rules on descent

```
PROCEDURE SORT_CHILD(bird A, bird B)
  IF A is TALL and B is SHORT:
    Return MEDIUM
  ELSE IF A is TALL and B is TALL:
    Return TALL
  ELSE IF A is MEDIUM and B is TALL:
    Return TALL
  ELSE IF A is MEDIUM and B is MEDIUM:
    Return MEDIUM
  ELSE IF A is MEDIUM and B is SHORT:
    Return SORT_BIRD(child of A and B)
```



5 Act as if groups deserve unequal treatment


<p>IF Bird is TALL or MEDIUM THEN:</p> <p>Flimsy sticks for nests</p> <p>Wait for worms</p>	<p>IF Bird is SHORT THEN:</p> <p>Best nests</p> <p>First dibs on worms</p> <p>Front of flock</p>
---	--

Figure 4.1: The worksheet for Birdcraft, depicting Hoyt's 5 steps of racialization as pseudocode.

Our “bird caste system” is algorithmic in structure, reflecting Hoyt’s five steps of racialization [206]. Hoyt, a diversity educator who speaks of disrupting racial ontology, teaches the process of racialization as an algorithm with five steps –Select, Sort, Attribute, Essentialize, Act [206]. Differences may not be visible but could be class-based or location-based, such as what someone owns (e.g., cattle was one indicator in Rwanda) or where someone lives; Wilkerson provides an example of height as a determinant of a fictional caste system [426]. Oppressors make the caste position hereditary through rules on descent, as in the Nazi Germany Nuremberg Laws, Indian caste system, U.S.-based one drop rule, blood quantum policies, or German and Italian citizenship laws [206]. Oppressors then act on these rules, seeking to naturalize them.

For instance, under Step 2: Sort we had the editable pseudocode:

2 Sort into group based on attribute (“leg size”)



```
PROCEDURE SORT_BIRD(bird):  
  IF bird's leg > 3 inches THEN:  
    Return TALL  
  ELSE:  
    Return SHORT
```

The algorithms are designed to sometimes lead to ambiguities or contradictions; e.g., you can’t use the Sort procedure to arrive at a Medium bird, this is decided only from ancestry –resembling how “mixed race” is determined in the U.S. [377, 142]. Reflecting this, in Step 4: Essentialize we had the pseudocode:

4

Essentialize differences: make caste hereditary through rules on descent

```

PROCEDURE SORT_CHILD(bird A, bird B)
  IF A is TALL and B is SHORT:
    Return MEDIUM
  ELSE IF A is TALL and B is TALL:
    Return TALL
  ELSE IF A is MEDIUM and B is TALL:
    Return TALL
  ELSE IF A is MEDIUM and B is MEDIUM:
    Return MEDIUM
  ELSE IF A is MEDIUM and B is SHORT:
    Return SORT_BIRD(child of A and B)

```



To introduce Birdcraft, we first explain Bird World, then **assign participants bird identities**: category (Short, Medium, Tall) and leg size (2–8 inches). We give some Short birds 3 inch legs (right on the sorting line) and some Tall birds 4 inch legs, to approximate the kind of messiness in real-world categories of race.¹⁸ Finally, **we task groups to make Bird World more “fair.”**¹⁹ Participants are put into breakout groups by their category and edit a shared worksheet (on the on-line version of this activity, a Google Slide). They could change the worksheet and pseudocode anyway they wanted. We wondered if and how

¹⁸“Short-passing” birds –analogous to white passing –emerge naturally as a consequence, leading to a few confused participants wondering why they were still considered “short,” given that their leg size put them right on the edge of being in the dominant group.

¹⁹I’d like to note how this design emerged from Ariam’s expertise. Originally, I had suggested a design where participants would be given a category –short or tall bird –and then tasked with attaining some material resources. This could have had the consequence of making it a competition, reproducing or entrenching the “heightism” of bird world. Instead, Ariam suggested that it be designed such that, regardless of their assigned category, participants were tasked to change the society for the better. This put participants in the position of a social activist, while also succeeding in making the “short birds” –the dominant group –a bit uncomfortable in reflecting on their status in this unequal society.

bird identities would play a role in their decisions.

Our design choice to give participants identities means that they could not operate from a “veil of ignorance” to produce just outcomes [346, p. 118], but rather reflected the position of critical race theorists, who argue that it is not possible for members of a society to disengage from their social positioning when working towards justice [115]. At the same time, the fictional framing of the activity aimed to help participants disengage from their real-world identities, while setting up a proxy world where discussions emerge that are analogous to real-world situations. The potential benefit of such a ‘proxy activity’ is that what could be a very charged discussion, were actors directly implicated in the debate (possibly spiralling into denial, defensiveness, or personal accusations), is diffused into a fictional situation, which nonetheless has analogous implications. The downside is that additional work may be necessary to transfer participants’ learning in the activity to the real-world context.

The Birdcraft activity ends with a short lecture introducing Morrison’s concept of the racial house, the notion that racial categories are like ‘rooms’ in the house, and connects it to three “stages” of anti-racism that I came up with: color-evasiveness, race-consciousness, and racecraft-consciousness.²⁰

²⁰Here are some definitions, paraphrased from an audio transcript of the lecture in question. In color-evasion phase, the people in the top room (in the U.S., wealthy white people) are saying, ‘well, everyone’s the same,’ and over-looking racism and inequality. While arguably better than explicit racism, this does nothing to resolve racial hierarchy. In the race-consciousness phase, one argues that we should “see race” and we should celebrate racial difference and strive for racial equality. Phrases are used like, “anti-racist ideas are those which declare racial groups are equals in all their apparent differences” (Kendi). Now, it’s like everyone’s in the living room eating together, and those from the top floors have pledged to refurbish all the other rooms so that everyone’s going to have the same quality room as everyone else. Ultimately, however, those in the top rooms are still in the (labelled) rooms, and if those below have kids, they’re not going to be able to get the keys to the top room. (i.e., racial groups in the U.S. are constructed and maintained through asymmetric cultural rules on descent; see Spencer, Ignatiev, and Wolfe [377, 211, 434]). Finally, the third stage of anti-racism is racecraft consciousness, which argues that we should see how we produce race. We should attend to racism, but also start to deconstruct race while we attend to racism. In a racecraft conscious world, at the

In the following section, I describe general takeaways from our two deployments of Birdcraft.

4.2.2 Deploying Birdcraft with participants

Ariam and I tried Birdcraft in two contexts: a small middle school class over Zoom (12 participants, including 2 adults), and as part of a larger workshop with CS educators (9 participants) at a prominent organization which facilitates professional development programs for CS teachers.²¹ Each took about an hour. The study was exploratory and informal: the former context was not recorded, and evidence is anecdotal (from notes written afterward). The latter was video recorded with the permission of the participants; however, because Breakout rooms in Zoom could not be recorded at the time the research was conducted, we could only capture the debate afterwards. Nevertheless, what little data was collected suggests the potential for cultural algorithms activities in the future. It appears that, as anticipated, groups came to very different conclusions on anti-racist strategy that are often analogous to real-world situations. While some of these analogous situations were by design, others were not anticipated by either of us, and emerged as a consequence of the setup.

Here I relay five situations that emerged: debates over whether the initial sorting is itself wrong (Steps 1 and 2), renaming of the oppressed group, dilemmas between deconstruction of caste and the need to reify caste in order to repair past harm, concern over the fate and status of “multiracial” individuals

end of the racial house, the racial house (the racial hierarchy) would be destroyed. (Arguably, the framing of this as “stages” conveys an ordering of value for each perspective, with racecraft as the most valuable; some may take issue with this ordering.)

²¹A professor at a state university also ran the activity, to anecdotal success, in his classroom on “Race and Technology.”

(Medium birds), and positioning multiracialism as the ‘solution’ to Bird World’s racism problem. When referring to groups, I will use a prefix to distinguish contexts: "MS" for middle schoolers, and "Educator" for CS educators. Note that in the middle school Zoom class, each group had one adult present.

Whether the initial sorting is wrong (Steps 1 and 2)

In both contexts, groups appeared split on whether the initial act of caste categorization was a problem. Some decided to keep castes but change the sorting procedure. For instance, the MS Mediums decided to categorize on Wing Span instead of Leg Size. They changed the Sorting algorithm to produce “Big” and “Small” categories, with the dividing line the average bird wing span according to Google. While they changed the Step 3: Attribute code to give all types of bird the “friendly” trait, they labelled Big birds as “assertive” and Short birds as “passive,” giving preference to a new caste of bird.

The MS Shorts (the oppressor caste) followed a similar tact. They decided to not change the categories or sorting rules, but rather distribute treatment more “equally.” Talls now got the best nests, but last choice on worms; Shorts the opposite; and Mediums flew at the front of the flock, but their nests and worms were just “okay.” This is akin to the “racial equality” route, where racial groups themselves are not problematized and the problem is distributing resources equally among them. Said the Educator Mediums, who took a similar route: *“We changed the traits because we were like that’s where it goes downhill. That’s where the negativity starts.”*

New algorithms for sorting also emerged for those who erased the original

sorting rules. For instance, MS Tall birds decided to erase caste categorization itself, arguing that the act of sorting was itself discriminatory. They removed selection and sorting procedures, but wanted to reassign jobs according to physical attributes. For instance, birds with large wings were flyers, birds with the sharpest beaks would collect food, and tall legged birds served as “workout trainers.” They defined no specific sorting policy for assigning categories.

Meanwhile, the Educator Short birds also decided to erase Select and Sort steps, but created more algorithms about what birds should do based on jobs they self-select into:

“We decided to we got nervous about the amount of resources on this island. I mean, there’s a lot of us here now, and we thought maybe we should just get rid of these attributes altogether... We wanted to know if people would be willing to split into groups of stick finders, food gatherers and nest builders. And then we wrote some algorithms to go with that. So where, you know, if we don’t want to split up into reasonably equal groups, then maybe we take turns doing things.”

They wrote their algorithms as Steps 1, 2, and 5 on the worksheet, after erasing the algorithms on traits and ancestry (Steps 3 and 4):

1

We believe that we need to work together to manage resources...

Group birds into:

- Stick finders
- Food gatherers
- Nest builders

Groups are porous - you can move between them. And you do what brings you joy.

2

Sort into group based on attribute (preference)

```
If each group has at least  
Population/5 members) {
```

```
Run RESOURCES}
```

```
Else{
```

```
Run Distribute Roles}
```

The RESOURCES procedure here was defined in Step 5:

5

```
RESOURCES
```

```
  If community needs food then {  
    FOOD GATHERERS gather worms;  
    Worm bin = full;  
    Everyone enjoys a meal together;  
    Worms are distributed to all birds equitably  
  }
```

```
  While birds in community need a nest {  
    If Stick_Bin = Full {  
      NEST BUILDERS build nest}  
    Else{STICK FINDERS gather sticks  
  }  
}
```

As this demonstrates, one emergent issue with the activity is that some groups could get fixated on the genetic attributes of the birds –how long legs might require bigger nests or could reach better food:

“Well, I know in our group [Educator Short birds], we started out with the conversation of, maybe the long legged birds can get like fruit off of a tree more easily and the short leg birds can focus on like... So we were still talking about collaboration, then [fellow Short bird] was like, but that’s still categorizing people based on this attribute. And so that’s when we went the other direction.”

This fixation with hereditary attributes also occurs above, when the MS Mediums recategorize on Wing Span and look up average bird wing size on Google to make their sorting algorithm “precise.”

Oppressed group renames themselves

In the activity with educators, the oppressed group renamed themselves, analogous to many such instances in history:

“We no longer identify as Tall bird. We identify as Big bird. And we changed the traits that were given to us. We thought long and hard about what we would do. And what we landed on is figuring out how to expand resources and become more adaptable... [Fellow Big bird], do you want to add to that?” “Sure. I just want to say, that as Big birds, we are beautiful, resourceful, smart, and also leaders. Y’all just didn’t know. But, because you came and joined us, we no longer have a lot of resources, so we thought of a few options. One, we want to work with you to figure out how to repopulate your land. Because you clearly came to our land for a reason. So, because we clearly have a better resource management strategy, we thought

we could share [that] with some of you. And we also thought of expanding our resources, like eating fish or bugs, maybe those are actually better."

Possibly, the educators in this group were using the artifice of the bird work as a proxy for talk about U.S. racism and colonialism (i.e., the group's choice to rename themselves could have been to *deliberately* parallel such acts in history). The reframing of Tall to Big caused participants to try to respectfully change how they called the other group (e.g., "*Our Medium birds change was re-assigning the trait of Big birds (previously Tall birds) to 'unique', nice, giving, and inventive as opposed to weird, scary...*"). The oppressor group tried to respect the oppressed group's decision, but also expressed a desire to "just be Birds":

"I mean, similar to the Tall –the Big birds, the Big Birds, sorry. [...] So [as Short birds], we're going to change our ways, and we're happy to call you Big birds. If we want to still refer to each other that way, or we can just, you know, we can just be Birds."

This last point is analogous to some white American's retort, when confronted with anti-racist efforts or assertions of Black or Indigenous identity: "why can't we all just get along?" or "why can't we all just be Humans?"

At one point Ariam asked the Big birds: "What if some of the Short birds refuse to accept your designation as being Big birds and they continue to just still call you Tall?" "*We just hang out with the cool birds,*" retorted a Big bird. "*Cool birds only! It's all fake, anyone who's cool can come hang out with us.*" At first glance, "it's all fake" here seems to relate to the constructed nature of the activity. In the context of the discussion, however, "it's all fake" is more likely

a critique of the sorting process itself (i.e., the same comment was made right after a comment about how two birds with same-length legs count as different categories of bird). In this reading, “cool birds” are those who have given up their dependence on caste sorting for their identity. However, note that the Big birds have *not* explicitly given up on calling themselves Big birds.

Dilemma between deconstruction of caste and the need to reify caste in order to repair past harm

In the activity with educators, a dilemma arose between deconstructionist tactics and the need to reify caste in order to repair past harm. After the Educator Short birds announced their solution was to erase caste categorization, the Big birds agree on principle, but question how quickly the Short birds want to move on from the past: *“There is a question about accountability, though. Because, cuz y’all [short birds] were real messed up... You know, it’s cool. You’re trying to change your ways. That’s real good. But where’s the accountability?”*

After this retort by the Big birds, a Short bird questioned their approach to consider accountability and reparations:

“What I was wondering about is like, where are the attributes enter the algorithm? And in the algorithm we were starting from, they were the primary kind of grouping. And so we deleted that and made those attributes irrelevant to the acquisition of materials, etc. But we could re-enter... we could do re-ascription or reattribution when it comes to the distribution of materials. But then there’s the leg length question versus like, is it actually tall or small? I’m sure many of you noticed there were like, tall birds

with five inch legs and medium birds with five inch legs. So like maybe we don't need that grouping, but we might be able to agree on some, like, need-realities based on specific attributes, i.e. create little mini algorithms that are in there... What I was thinking about is... how can an algorithm register difference in equitable ways without producing that difference as a 'reality effect' that then everything else is based on and seems immutable?"

This question of “who gets reparations” reflects real-world debates on how to determine and best enact reparations for the long legacy of racism in the U.S. As Spencer and Hoyt argue from the deconstructionist anti-racist position [377, 206], some ascription is necessary in the short-term for redressing past harm and reallocating resources. The question “is it actually tall or small?” showed the inherent difficulty of relying on racial categories, given their constructed and inconsistent nature.

Concern with fate and status of Medium (multiracial) birds

In both contexts, the fact that some birds with same-size legs were sorted into different castes –a deliberate design choice by us, that emerges from the arbitrariness of the Bird world cultural algorithms –bothered participants. For instance, an Educator Medium bird asked another “how tall are you? How long are your legs?” to which they replied five inches. A Big bird remarked, “[Big bird] is also five inches. So [other Medium bird] and [Big bird] have the same length of legs.” I added to this, remarking that I have only three inch legs (I was given a Medium bird identity, but did not take part in their breakout room discussion). Said a Medium bird: “We couldn't figure out how these three very different birds ended up in this category.”

Identifying inconsistencies in the caste categorization algorithms intersected with a concern over the fate of Medium and “passing” birds (Short birds with 3-inch legs). For instance, after the Educator “Big” birds proposed to repopulate the Island where the Short birds came from, a Medium bird interjected, “well if we’re gonna repopulate the place where the Short birds came from, where are the Medium birds supposed to go? Because we’re not short or big, but some of us are kind of big and some of us are kind of short.” In the MS Shorts, one bird with 3 inch legs argued against changing the rules because they were “just squeaking by” to sort into the Short group. This is analogous to a situation where an American who passes as white is reluctant to give up the privileges afforded by whiteness, at the expense of those who cannot pass.

Multiracialism as the ‘solution’

In the middle school class, two groups positioned multiracialism as a solution to the problem. Short birds changed the ancestry algorithm to allow the offspring of Tall and Medium birds to be Medium instead of Tall, and justified the choice by claiming, “it’s more fair if more birds can be Medium.” The Medium birds meanwhile changed the sorting algorithm on descent, adding new categories “Medium big” and “Medium small” for describing offspring. One member’s rationale was that more Medium categories might diffuse discrimination.

Positioning multiracial identity as the solution to racism resembles the multiracial identity movement of the 90s [377]. Spencer argues that such a position merely serves as yet another smokescreen for racism in the U.S. and supports white supremacy. He argues that there is nothing novel about “multiracial” identity, as across the history of the U.S., there were many (now outdated) terms

for “multiracial” individuals. Multiracialism hides anti-Blackness, in that it implicitly normalizes the premise that there are pure races and obscures the cultural one-drop rule against Afrodescendants in the U.S. The only solution, he argued, was for those in the U.S. to practice what he called *racial suicide*, or opposing the notion of racial identity.²²

4.2.3 Reflections

After the activity, CS educators expressed that they wanted more time to reflect on how their strategies and debates connect back to real-world structures and debates in anti-racism. In particular, one participant commented that they found the Morrison metaphor and the color-evasiveness, race-consciousness, and racecraft-consciousness framing helpful to situating their debates (types of anti-racism), but would have appreciated more explicit connections to back to their choices during the activity. We ran a follow-up session with the educator group over a month later. Educators expressed that they wanted more Cultural Algorithms activities in other areas, including gender identity and social-emotional learning, such that race/ism was but one activity. Inside this unit, Birdcraft would appear later on, once students had grasp of the ‘cultural algorithms’ concept. Second, they wanted a facilitation guide along with the activities, including example responses from students and how to connect these responses to real-world debates and history.

²²“It is always simply assumed that people need a primary racial identity in order to be complete human beings... The blame for this false consciousness can be laid squarely at the feet of both psychology and sociology... Race is not a social reality in the U.S.; rather, fallacious belief in race is the U.S. social reality... Unlike the social constructionist, I do not argue that people should accept and adopt racial identities; indeed, I argue quite explicitly that they should not.” [377, p. 249-64]

While overall the activity seemed generative to participants, there were some responses and limitations to the activity that future “cultural algorithms” activity designers should be cautious of. First, while it can be useful to describe a *situation* in algorithmic terms, our algorithmic framing may have caused some participants to look for and describe the *solution* also in algorithmic terms. Participants may then look for a clean, top-down, immediately implementable solution, rather than one focused on mobilizing others and building community. Second, our anthropomorphic framing seemed to lend itself to further essentialism of subjects, such as groups categorizing birds based on wing span or associating birds with longer beaks as better resource grabbers. Other birds without these normative characteristics might then be implicitly stigmatized, producing another —albeit different —social hierarchy. A further iteration of this activity might call on participants to question such ideas, including the limitations of the algorithmic framing itself.

Reflecting on the activity worksheet, I would make one alteration. Educator participants wanted to know how the dominant caste came to be seen as having positive traits, while non-dominant castes were perceived as having negative ones. Hoyt’s racialization steps have Attribute as Step 3. However, a key point of *Racecraft* is that attributes are normalized only *after* the oppressor acts upon caste categorization over long periods of time. Placing the “Attribute” step after the “Act” step would clarify this directionality.

4.3 Conclusion

In the future, I imagine Birdcraft sitting within a longer unit on “Cultural Algorithms.” The goal would be to get students to see discrimination and societal bias as *actions*, rather than static, pre-given, unchangeable facts. However, such activities would need to help people transfer their critical thinking about cultural algorithms in fictional contexts onto analogous real-world structures. Following this, we could lead into project-based activities to address “what we do” with this knowledge.

For many of us, it might be terrifying to imagine students remixing caste systems through algorithmic concepts. But tinkering with oppressive systems to understand how the hidden gears within them work is one way CS education might support students to understand systemic discrimination. This does not mean living in a colorblind world. Quite the opposite: it means acknowledging and understanding how we have been programmed to assign value to different groups of people, driven by capitalism and exploitation. It means striving for Morrison’s *home*: a future that invalidates the racist paradigm as it is expressed in our laws, schools, and daily life [286]. Activities like Birdcraft might sound like the start of a piece of speculative fiction written by the legendary Octavia Butler —and that’s no accident. Properly and respectfully framed, innovative work around visionary fiction and social justice can provide us with the imaginative resources to realize better futures with and through computing concepts.

Part II

Culture in programming

CHAPTER 5

THE ORIGINS OF PROGRAMMING AS CULTURAL ACTIVITY

“History, as nearly no one seems to know, is not merely something to be read. And it does not refer merely, or even principally, to the past. On the contrary, the great force of history comes from the fact that we carry it within us, are unconsciously controlled by it in many ways, and history is literally present in all that we do. It could scarcely be otherwise, since it is to history that we owe our frames of reference, our identities, and our aspirations.”

—James Baldwin, in “Unnameable Objects...” [38]

The former chapters explored an intercultural perspective on programming education that addressed the question: how can programming concepts and activities be put towards people’s intercultural development? Aspects of this question were answered, but much more work needs to be done. What about other “cultural algorithms”? How do we measure students’ social and cultural learning? These questions are all important, and deserve time and attention. Nevertheless, as I carried out this educational work, another, very different question gnawed at me.

How are tools of programming themselves “cultural” artifacts? Were not the artifacts and techniques of classrooms also “cultural”? What assumptions and values might they circulate? I started to question the material over which people collaborate —plaintext editors, the QWERTY keyboard, the mouse —as representing the culmination of historical processes. What exactly is the heritage of Scratch, of monitors and keyboards and programming “languages”? After searching far and wide, however, I could not find a satisfactory account

that was broad enough to bridge the literature on STS, HCI and programming, without becoming either a detailed historical piece surveying just a tiny part of history, or a piece that anachronistically inserts present-day terms, overlooking how today's commonsense terms and phrases used to describe programming have their own history as social constructs.

To help fill this gap, this chapter traces the sociomaterial fabrication of early computer programming notation and practice: of how 'to write code' came to imply typing characters in text editors and terminals, rather than (for example) a practice involving handwriting or drawing. Through three case studies of the earliest visions of 'writing code,' I recall the emergence of high-level¹ programming notation and computing's extension of earlier social and material infrastructure of the typewriter and card-based processing. Adopting a cultural-historical sensibility to technological emergence, I trace how early inventors' practices in logic, physics, mathematics, engineering, and art –with their varying, handwritten and drawn notations and sensibilities –informed and directed how they fabricated programming notations and practice. I argue that an initial diversity of styles quickly came into conflict with typewriters and card-based pipelines, and that notations were thereby serialized and transformed. Corresponding to this transformation and coordinated with the need to align computer science with formal abstraction to justify the new discipline [396, 92], values became attached to different forms of programming notations that reflect prior modernist dichotomies in North American and European societies between the 'textual' and the 'visual,' the 'written' and the 'drawn' [214], aligning the former with objective authority and the latter with aesthetic choice [318]. By

¹High-level refers to notations that would need to be considerably interpreted into machine code (numbers) in order to run [241]. Assembly code largely serves as a mnemonic device to numeric codes and are excluded.

tracing this history, I (provocatively) redefine programming as not just involving the design of algorithms or systems, but often “a problem of mapping from one culture to another” [109, p. 138]—from its very inception, a practice which so often involves *translation work* which is intimately tied to intercultural conflict, compromise, and innovation. In so doing, I advance an intercultural lens [215] on programming practice as a site of social, material, and epistemological contestation, not just in the design of contemporary software or for those outside the societies where computing emerged, but embedded in the design and history of the very tools and practices which support the development of software, and the communities and discourses which have formed around them. In the process, I join other scholars [14, 56, 261] in seeking to “move the centre” [417] in discourse around programming, making, and HCI more broadly towards a plurality of cultural perspectives and practices, while leaning away from overly simplistic rhetoric of the ‘West’ that denies inner heterogeneity.

To begin, I contend with why ‘writing code’ deserves attention, when there have been numerous attempts—such as tangible programming or direct manipulation languages [278, 209]—to reconstitute programming practice. To demonstrate how even the most radical visions of programming can end up recentring a typewritten status quo, I offer the following contemporary anecdote.

5.1 The Paradox of Change in HCI

In 2017, a group of artists, designers, and software engineers, led by former Apple designer Bret Victor, came together in Oakland, California to forge a new vision for the future of computer programming. Their goal was “to incubate a

humane dynamic medium whose full power is accessible to all people,” pledging to liberate professional programming practice from the confines of stuffy offices, isolating screens, and constraining devices. “No screens, no devices” became the project’s motto and organizing principle. No longer impersonal and individualized, programming would become communal, social, and democratized. People would “think with their hands, their bodies, spread out, walk around, compare possibilities, improvise, and experiment” [413].

DynamicLand is a room-sized operating system that realizes visions from ubiquitous computing and tangible programming, an impressive achievement by any measure. Yet despite the refrain of “no screens, no devices,” DynamicLand’s core infrastructure is Lua code printed on pieces of paper. To make alterations to the room-sized program, programmers regularly re-constitute the bottleneck of the shift-key keyboard and associated standards of ASCII symbols laid out in left-to-right, top-to-bottom sequence on a screen [372]. In order for “all people” to gain access to computing’s full power, to liberate themselves from all the screens and devices, the bottleneck of the screen and keyboard again re-formed.

From a broader vantage point, the DynamicLand paradox highlights a growing discomfort with contradictions between utopian rhetoric and technology’s actual ability to enact change, whether in education, politics, international development, organizations, or design [403, 330, 261, 358, 381, 311]. In HCI and beyond, wide swaths of people are now wary of failed promises. Contradictions abound: companies who profit off of user attention release tools to monitor attention; the rich, after buying the newest marginal phone upgrade, pay again to have it taken away [208]; teenagers, whose computer use we are told we should

be concerned about, protest computer over-exposure [384]; the same billionaires that send their children to ‘disconnected’ schools pour millions into connected learning programs [403]. As the anthropologist Alexei Yurchak found of the ailing Soviet Union, tech today is “simultaneously eternal and stagnating, vigorous and ailing, bleak and full of promise” [442]. Alongside this growing disillusionment with computing technology is a corresponding de-mystification of its design and professionalization processes. As the field of science and technology studies (STS) has shown, ostensibly technical disciplines are replete with social elements: technologies are socially constructed, co-produced with society, and value-laden [57, 432, 219, 7]. STS perspectives have shed light on programming practice and histories, whether the role of trust and professional vision in data science [318, 317], the marginalization of weavers as programmers in the Apollo program [358], the construction of computing as a science [396, 7, 109], or the framing of coding as a literacy [410]. Postcolonial scholars have also attempted to decentre dominant narratives in the maker movement’s rhetoric [261, 14], suggesting that maker practices should be more inclusively framed as “making do”: “using the materials and competencies on hand to create objects or processes that aid in everyday life,” rather than framed as inherently revolutionary or democratizing [14].

A complementary issue to the goal of decentring dominant narratives is how the tools developed to support programming condition thoughts and imaginations. Past scholarship on the influence of material representations on knowledge construction argues that the structure of (typewritten) notations influence the kinds of problems encountered in their usage [122, p. 8-9]. This argument is suggestive of situated theories of cognition such as cultural-historical activity

theory (CHAT), widely applied in CSCW and CSCL² and deriving from Russian psychologists Vygotsky and Leont'ev [228, 94]. A CHAT perspective emphasizes the cultural³ and historical roots of social activity and its mediators and argues that learning relies on gradual 'internalizations' of 'externalized' cultural tools ("such as algebraic notation, a map, or a blueprint" [228, p. 42]). The term culture here operates in a generative, rather than a taxonomic sense, and is defined by Irani & Dourish as "a lens through which people collectively encounter the world, a system of interpretive signification which renders the world intersubjectively meaningful... [A]n individual may participate in many cultures –cultures of ethnicity, nationhood, profession, class, gender, kinship, and history –each of which, with their logics and narratives, frame the experience of everyday life" [215, p. 2-3]. From these perspectives, programming notations are cultural tools, affording certain thoughts and frames while resisting others.

However, tools and practices for new disciplines like computing do not arise in a vacuum; rather, situated practice extends and appropriates pre-existing culture to new ends. As Pickering argued of physicists, this extension operates through a mangle of practice, a dialectic of resistance and accommodation between humans and machines [332]. Comments in postcolonial-oriented papers can cast programming languages as arbiters of a Western monoculture (e.g., [56, 174]), and while reflecting on the entrenchment of certain values, representations, and assumptions is useful⁴ –in the spirit of Ong's contrasting of written and oral societies [310] –it is perhaps too simplistic an argument to accom-

²Computer-Supported Cooperative Work and Learning, respectively.

³Throughout, I use the term "culture" in a broad sense following CHAT approaches, Andrew Pickering's concept of the mangle [332], and Irani & Dourish's postcolonial interculturality [215], rather than as a taxonomic classification attached to geographic boundaries.

⁴Importantly, groups outside of Anglo societies have raised these concerns, such as indigenous Hawaiians converting C# to their language [291], Nasser's Arabic programming language Qalb [295], and Nguyen's account of a Vietnamese software community [14].

modate inner heterogeneity. As Dourish notes, in order to “get a grip” on the cultural consequences of technology, we must take seriously its “material specificities” rather than speaking of an amorphous and unexamined presence [121]. What exactly is the cultural heritage of programming notation and practice, beyond the obvious use of English keywords? How did ready-to-hand cultural practices influence the design and development of new ones? And how might the spread and influence of early approaches impact the current field, whether visibly or in deeply held, almost invisible ways? While not claiming to answer these questions definitively, this work aligns itself with a growing number of scholars at CHI and beyond arguing for deeper engagement with HCI’s early history [358, 359, 237, 8, 14]. I build on this work by trying, as much as possible, to avoid casting our present-day assumptions onto the earliest history of programming. Unlike other work on programming focused on end-users, for novices, or otherwise tools or studies to support the typewritten status quo, I call into question here the entrenchment of the dominant regime.

5.2 Human-Machine Interaction Before HCI

To orient ourselves in the past, I briefly outline the state of “human-machine interaction” around the advent of digital computers. Some historians now regard the emergence of digital computing as a period of gradual change, rather than a revolutionary discontinuity [109, 181, 101]. The machine ‘computer’ extended workflows of industrial data processing at a time when writing was being disassociated from writing ‘by hand’ and increasingly associated with writing through a discretizing (or “technolinguistic” [290]) mediator [101, 28, 214]. The history of HCI is thus also a history of the “eternal recurrence” [239] of the

typewriter, and to this technology I shed some light here.

The dominance of typing is of course a topic many scholars have commented on. Like Latour's crashing of distinctions between the human and nonhuman [248], early twentieth century philosophers framed typewriters as having agency. Nietzsche wrote that typewriters have "fine fingers, to use us," and Heidegger, who believed the hand was "the essential distinction of man," warned that the typewriter "imposes its own use" on humankind, transforming "the relation of Being to man" [239, p. 198-200, 207]. Phenomenologists and cognitive scientists would echo these notions later in their concerns about, for instance, shifts from the level of "paragraph" to "sentence" and shortening attention spans [266, 81]. Recently, the anthropologist Tim Ingold harkened back to these concerns when he called on HCI designers to imagine an antidote to discretizing technology: "a technologically enhanced sensitivity, brought into the service of hands-on engagement with materials in making, [which] could genuinely enlarge the scope of humanity, rather than further eroding it" [213, p. 124].

The history of the typewriter in some respects reflects warnings about its homogenizing effects. While much has been made (and debated) over the QWERTY layout [42, 110], a less reflected on and even more pernicious case of path dependency is the shift-key mechanism. The Remington II shift-key typewriter was optimized for the lower frequency of capitalized letters in English and first designed by American companies embedded in a left-to-right, top-to-bottom written culture [7]. In *The Chinese Typewriter*, Mullaney shows how non-English writing systems were adapted to the standard with as little modifications as possible in order to lower factory assembly costs. In popular media, the type-

writer became a symbol of modernity, a machine whose dissemination brought the coming of civilization. Written cultures became judged by their ability to be consumed by the shift-key style and faced pressures to change or romanize. For instance, Chinese logograms were debased in both global and domestic discourse (e.g., by Mao Zedong), and the Thai language lost two characters due to space limitations (a change which persists today). Moreover, the name ‘typewriter’ conditioned how people interpreted and imagined other technolinguistic machines; for instance, early Chinese typewriters did not in fact have keys [290].

As the shift-key typewriter monopolized writing and threatened cultural diversity, an intimate symbiosis between the computer and the typewriter became drawn in the theories and metaphors that retroactively [180] came to define the discipline of computing. As a boy, Alan Turing pictured himself inventing typewriters to mitigate his poor handwriting; as an adult he preferred to type than write [239, 200], behavior atypical among many of his mathematical contemporaries.⁵ In his seminal 1937 paper on computability, Turing describes a generalized typewriter that prints characters on a page with four extensions: it uses an infinite paper tape; can remember and erase symbols in place; and, inspired by the shift-key, may switch between a variable number of configurations [406, 200]. Turing’s later paper on artificial intelligence calls the typewriter interface the “ideal arrangement” through which to verify intelligence, echoing the civilizing guise of earlier discourse. At the same time, infrastructures of large-scale data processing came to prominence in U.S. accounting. Dorr Felt’s Comp-tometer and William Burroughs’ adding machine products, fitted with keys inspired by typewriters, were “the two most popular sets of devices available in

⁵As was common at the time, Turing would handwrite in mathematical notation that extended beyond his typewriters’ abilities (e.g. [200, p. 356]).

the U.S. at the turn of the [20th] century” and could add and multiply numbers [28, p. 30-1]. For larger businesses or government data processing, Herman Hollerith’s punch card machines formed a system for storing, tabulating, and sorting data [109] (prominent manufacturers included Hollerith’s International Business Machines (IBM) and Remington Rand). As Aspray notes, these three “legs” of computing –typewriters, adders, and punch card processing –were already stabilized for around three decades before the advent of electronic computers [28]. The “operators” of these machines –typists, clerks, secretaries –were largely and increasingly women. Only the advent of World War II brought additional attention on applied mathematics and temporarily destabilized gender roles in the data processing industry [7, 385]. By the advent of digital computers, the social elements of computing –routinization and the division of labour, and feminization of the workforce –had long been in place [243].

5.3 The Culture ‘in’ Early Programming Notations: Three Visions

According to a report by Knuth & Pardos [241], the two first high-level programming notations emerged around the mid-1940s in Germany and at the ENIAC project in the United States. Several years later emerged several typewritten visions of programming in Europe and North America. Here I explore these earliest acts of HCI design through lenses of materiality and situated, cultural perspectives on knowledge construction.

5.3.1 Konrad Zuse's Vision

During the cold opening months of 1945, as Allied planes bombed Berlin, the German inventor Konrad Zuse huddled in safety with his family. His inventions the Z1-3 computers lay blown to pieces in the ruins of his center-city workshop. He secured a truck to transport his wife, assistants, and sole remaining computer, the several-ton Z4, out of the city. Reaching the alpine village of Hinterstein over 650km south, Zuse setup the Z4 in a barn, but found the machine was broken. Secluded from the world with no means to continue construction on computers, he put pen to page, seeking a "universal formula language" for computation as an extension of his dissertation [450, p. 212]. This language he named the Plankalkül, the first high-level programming notation [241, 181].

All of Zuse's prior training and interests accompanied him to Hinterstein. As a photographer and artist, he had painted posters, wrote poetry, and acted and directed theatre, performing as "unknown inventors or artists" [450, p. 26]; as an engineer and inventor, he was trained in mathematics, formal logic, and civil engineering. He combined these seemingly disparate interests in efforts such as applying descriptive geometry to the optimal viewing of artistic work, or using punch cards to automate photography dark room processes [450, p. 17-19, 28]. Zuse seemed to relish the benefits technology of writing entailed, and became agitated when they were suppressed. In college, he switched majors two times in upset over drafting classes, stating that they "had shattered my illusions. The creative spirit was left little freedom in the manner of presentation; everything was standardized, everything was decided: the line thickness, manner of dimensioning, even the positioning of dimension figures" [450, p. 15]. While living in Hinterstein, he continued practicing art by engraving scenes

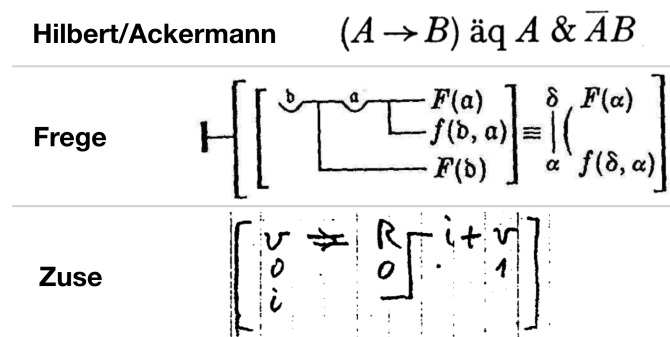


Figure 5.1: Excerpted notations of German logicians Hilbert/Ackermann and Frege, whose works Zuse studied. Below, a handwritten excerpt from Zuse’s 1945 Plankalkül notebook, showcasing the *Zeilenverschiebung*, or ‘line shift’ notation [448]. *From original texts* [198, 147, 449].

into wood blocks.

In creating the Plankalkül, Zuse drew from his artistic disposition and passionate interest in formal logic. Of the former, he never appeared to question coding as involving drawn lines and written notation. Of the latter, he dreamt of the universe as “a giant computing machine” and became obsessed with visions of a “mechanical brain,” what today we would call general artificial intelligence. This analytic brand of philosophy he likely inherited from German logicians whose works he studied closely, notably David Hilbert, Wilhelm Ackermann, and Gottlob Frege [450, p. 44-6,83,105]. Acknowledging the limits of numeric computation, he designed his ‘calculus’ around propositional and predicate logic in order to solve chess problems [241].

Frege’s work deserves some comment here. European mathematicians at this time preferred the aesthetics of linear sequences, and, if they were aware of Frege’s work, frowned upon his two-dimensional notation (Figure 5.1). For instance, historian Florian Cajori called it “repulsive” [72] and logician Ernst Schröder “ridiculed” it as Japanese [279], hinting at xenophobic underpinnings

behind some aesthetic judgements. But Zuse did not seem deterred. What today would be considered a single “line” of code is expressed across three rows as a matter of routine: the first row commonly including variables, the second denoting subscripts, and the third types [356]. Zuse remarks on the ease of “drawing a line” across multiple rows of characters to connect indices and liberally adopted notation⁶ from mathematics and formal logic [450, p. 219]. Later scholars, in reflecting on the Plankalkül, commented that the notation was “clumsy,” “unorthodox,” and puzzling [44, 356], reflecting a similar aesthetic valuation as that held against Frege.

From a technical standpoint, Zuse’s Plankalkül contained what would be considered “standard features” in approaches over a decade later and embodied a functional programming style twenty-five years before similar developments occurred in the Anglocentric community [151, 44]. When John Backus gave a Turing Award Lecture on functional style in 1977, he lamented the entrenchment of the imperative paradigm he helped create, calling numeric languages “conventional” and arguing that they had to be “liberated” from a “fixation” on the von Neumann computer [32, p.616]. Ironically, Zuse had built a similar functional style into his language around the same time von Neumann was handwriting the *First Report on the EDVAC* [415] which came to define what a “von Neumann computer” is. Zuse wrote of the times, “as a German it would have been difficult to gain the necessary attention at discussions dominated by Americans” [450, p. 128].⁷

⁶Notation included the square root $\sqrt{}$, power n , times \times , infinity ∞ , dot \cdot , delta Δ , Greek letters σ , ϕ , τ , ε ; logic notation included \wedge and \vee , open arrow \Rightarrow , and overline \bar{c} as negation [356]. These last four are likely from Hilbert/Ackermann [198].

⁷While Zuse’s work was only published almost two decades later, Rutishauser, Böhm, and the British and French governments appeared aware of it [450, 60, 364].

5.3.2 The ENIAC Vision

A similar vision of computer programming as involving written and drawn coding emerged around the same time in the United States on the ENIAC project, although for different reasons. The ENIAC project is of course well-trodden territory in the history of computing, considered a landmark for the electronic stored-program [182, 7, 258]. The women of the project, for decades dismissed as “operators,” gained belated recognition [43]. This section focuses on the cultural influences behind the written practices of the project’s vision of high-level coding, formally published in the 1947 *Planning and Coding Reports* by Herman Goldstine & John von Neumann (hereafter GvN) with the aid of Adele Goldstine and Arthur Burks [164, 163]. These reports were the first to publicly formalize computer programming as a methodology and popularized the term “programming” [181].

The ENIAC computer was built and run in the Moore School of Electrical Engineering at the University of Pennsylvania between 1943-55. In the early stages of development, ENIAC programs were entirely represented as a sequence of machine operations (“machine code” or, in their terminology, order codes), then painstakingly converted to switch flips, plug-board arrangements, and punch cards by women who physically programmed the machines [182]. Deciphering meaning from sequences of orders was extremely difficult. Von Neumann had worked out sorting algorithms in detail, and so had intimate knowledge of the challenges facing the translation task. The first merge sort algorithm he wrote by hand [414] which was common; his handwritten reports often had to be typed up by others [341, p. 6]. To cope with the growing complexity of machine-level coding, GvN developed a notation of box-and-arrow diagrams they called “flow

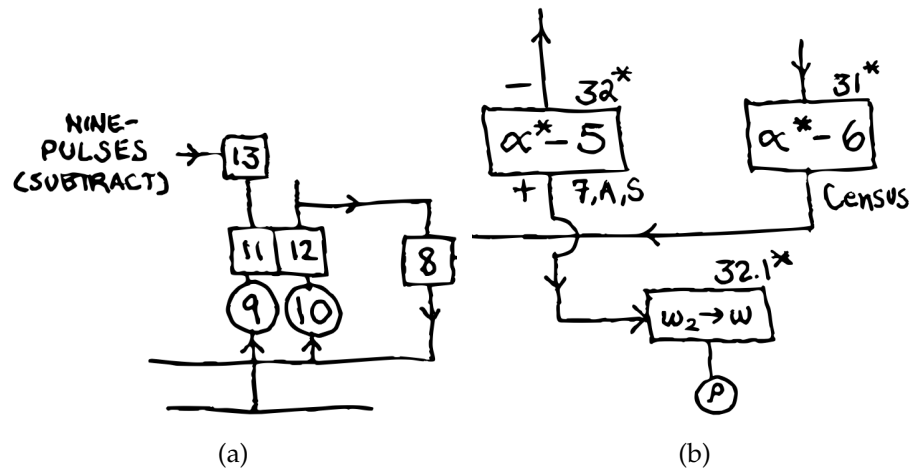


Figure 5.2: (a) Section of ENIAC accumulator block diagram abstracting an electronic circuit, Arthur Burks, Aug. 1947; (b) Section of flow diagram hand-drawn by Adele Goldstine, Dec. 1947. *Rewritten from originals* [70, 162].

diagrams.” In the report, GvN state that “coding begins with the drawing of the flow diagrams.” Such was the “dynamic or macroscopic stage of coding” [164, p. 20]. Although later work in software would characterize drawing diagrams as purely documentation, extraneous to the practice of coding and “drawn after, not before, writing” programs [65, p. 194], coding in the ENIAC vision was firstly inscribing by disciplined hand. In later stages of “static coding,” equations in boxes were converted into machine code and substituted for numbers. The four stages of preparation reflected the division of labour established in the data processing industry decades earlier [340].

The written culture of the Moore School’s electrical engineers was central to how GvN conceived of this practice (Figure 5.2). GvN originally used the engineering term “block diagram” in an early draft of the *Planning and Coding* reports [341]. Block diagram had been terminology used in electrical engineering for at least a decade prior [285, 54] and similar diagrams were drawn by hydrodynamic engineers and in industrial manufacturing [133, p. 326]. Block

diagrams were commonplace in the Moore School, as may be seen in the initial proposal for the ENIAC machine itself in spring 1943 [7, p. 90] and a later paper by Burks [70]. The notation gave an abstract picture of a circuit or system. Rather than including all the fine details in one picture, the block diagram would “block” out sections with labelled boxes, to be filled in later with more detailed figures. Flow diagrams did similarly, except mathematical operations would be written inside the boxes. The similarities between notations go beyond from the mere use of blocks and connecting lines, however, and include + and – beside conditional blocks (taken from polarity in electrical schematics), directional arrows, and semi-circles to ‘hop’ over visually intersecting lines [182]. When the flow diagrams were later in use, “the interaction of mathematicians and computer operators, among others, created a pidgin version” of the notation as it met the realities faced by implementation [341, p. 88].

Yet other factors appeared to contribute to GvN’s choice of notation. GvN were familiar with formal logic and abstract mathematics, so they arguably had the technical ability to approximate Zuse’s vision. However, several factors worked against this possibility. First, their institutional directive was to calculate equations, not solve logic conjectures. Second, von Neumann by this time had grown disillusioned with formal logic [7] and the ENIAC programmers were unfamiliar with it [202]. Third, the ENIAC administrators had to coordinate between a large staff and a rigid, secretive organizational hierarchy [7, 133]. It is thus likely that GvN believed that choosing a representation Moore School members were familiar with was superior to introducing a more radical departure in notation. The remarkable historical work of Priestley [341] unearths similar reasoning in a first draft of the *Planning and Coding* reports:

“[W]e have acquired a conviction that this programming is best accomplished with the help of some graphical representation of the problem. We have attempted... to standardize upon a graphical notation... in the hope that [it] would be sufficiently explicit to make quite clear to a relatively unskilled operator the general outline of the procedure. We further hope that from such a block-diagram the operator will be able with ease to carry out a complete coding of a problem.” (p. 59)

Note here the assumption that the “unskilled operator” –the women –could easily follow a notation rooted in the culture of electrical engineering. This was not true at the start of the project. Unlike members of the Moore School, the six ENIAC programmers (formerly human computers and mathematicians) had no training in electrical engineering. Upon arrival, they were merely handed block diagrams and asked to study them.⁸ Jennings Bartik recounts, “I had never read a block diagram in my life. Betty [Snyder] hadn’t either, and we assumed it was read from left to right like a book... I am still amazed at how little help, instruction, or supervision we had” [43, p. 75, 80]. This suggests that Bartik & Snyder applied their existing cultural knowledge to the new material, making sense of the situation as best they could.

Flow diagramming later spread largely due to von Neumann’s celebrated status [7, 285], rather than any concerted effort by a group or individual. Yet in the 1950s, Saul Gorn of the Moore-affiliated Ballistics Research Lab [7, 302] would make explicit the move to universalize the method. Gorn founded a re-

⁸After extensive archival work, Haigh *et al.* conclude that programmers’ later block and flow diagramming methods “were based on work done long before they were hired,” which corrects some prior accounts [182, p. 95].

search programme to find a “Universal Code” where “the flow chart would be the code” to be translated later into a representation suitable for any underlying machine [222, p. 20]. This argument influenced the later ALGOL international commission, a (largely failed, but highly influential) North American and European effort to standardize a universal notation to define algorithms [396]. Ultimately, flow diagramming would have a powerful influence on programming and software engineering for decades to come, becoming the basis for the “visual” paradigm of coding [62, 285] –a point to which I will return.

5.3.3 The Typewritten Vision and the Serialization of Programming Notation

From 1950 onward, visions of typing (non-numeric) symbols to program emerged, beginning with assembly code [241, 222]. Prior historical work on writing and programming often begins with this era (e.g., see [410, 302, 9]). Unlike the relative isolation of the prior two visions, by this time a computing community began to form [7], making it more difficult to trace influences.⁹ Both Swiss mathematician Heinz Rutishauser and Italian student Corraldo Böhm envisioned use of a keyboard years before the more well-known MIT WHIRLWIND and IBM FORTRAN projects [364, 60]. Here I touch on the work of Böhm, at MIT and at IBM.

At a high level, and more rigidly than the prior two visions, typewritten approaches were fabricated through a “dialectic of resistance and accomodation”

⁹For example, from 1948-9 Swiss mathematicians Eduard Stiefel and Heinz Rutishauser visited von Neumann, returned to Europe and met Zuse, his Z4 and Plankalkül –effectively bridging the isolation between visions. In 1952, they helped develop the Swiss computer ERMETH [299].

$$\begin{aligned}
 & \mathbf{x(i) = b \times \times i} \\
 & \mathbf{sqrt(a + b) : means \sqrt{a + b}} \\
 & \mathbf{a(i(j)) : means a_{i_j}} \\
 & \mathbf{a(i) = a(i) + 5.1 \times \text{sum}(j, 1, 20, b(i, j) \times c(j))} \\
 & \quad \curvearrowright \mathbf{a_i = a_i + 5.1 \times \sum_{j=1}^{20} b_{i,j} \times c_j}
 \end{aligned}$$

Figure 5.3: Examples of translations from mathematical notation to what FORTRAN designers anticipated could be typed on an IBM keypunch (1954). Notice the handwritten \times symbols and lowercase letters: by 1956, these became asterisks and uppercase letters [34, 35]. *Rewritten from original* [36].

between humans and machines [332, p. 22]. In this dialectic, machines resisted cultural practices which had developed along different material constraints. In turn, inventors accommodated the resistance through workarounds or modifications. The degree of accommodation depended on *what* specific machines inventors had on-hand, whether they actually implemented their vision, and the flexibility of the organization (if any) they operated under. A strong commonality between the inventors of typewritten visions was how they designed for mathematical users.

Examples abound of the resulting accommodations. In early 1953 at MIT Project WHIRLWIND [144], Laning & Zierler took a Flexowriter teleprinter (a combined manual keypunch and punch-card controlled printer whose design had passed through IBM) and began development of an “interpretive program” that they optimistically describe as being able to “accept algebraic equations... At IBM, FORTRAN designers had less flexibility to alter machines to suit their needs. John Backus, who founded the effort, was a college-educated mathematician hired to calculate Fourier series; long hours and difficulties of using ma-

chine code motivated him to “make it a little easier” [61]. He submitted a proposal to IBM management to lead a team to expand on this idea [33]. Operating in relative obscurity due to political tensions between Thomas Watson Jr. and Sr., the FORTRAN team had to work within the constraints of IBM’s ecosystem of standardized calculating machines built for business, aircraft, and government markets. They made do with IBM keypunch limitations by, for instance, using parentheses to denote super- and sub-scripts, adopting * and ** for multiplication \times and exponentiation, and enforcing all uppercase letters. Though these changes may be seen as concessions, they may also be seen as standards enforcing an economy of notation rather than idiosyncrasy.¹⁰ Yet for typewritten visions, the material form of the machines enforced constraints that went beyond mere symbol swapping. The linearity and limited size of punch cards and the left-to-right, top-to-bottom norm of Anglo- and European societies enforced a notation that involved a sequential series of horizontal rows of characters, where the number of characters was limited by punch card size. Semi-sequential notations like $\sum_{j=1}^n$ confounded serial input (i.e. it is unclear whether n or $i = 1$ came first), and thus had to be serialized e.g. `SUM(J, 1, N, . . .)` [37, p. 10]. Backus & Herrick’s IBM Speedcoding paper in 1954 describes the challenges facing the translation task between “rich” mathematical notation into “fairly involved” typed expansions:

“Obviously the programmer would like to write... ‘ $X + Y$ ’ instead of: ‘CLEAR AND ADD 100’... To go a step further he would like to write $\sum a_{ij} \cdot b_{jk}$ instead of the fairly involved set of instructions corresponding to this expression.” [36, p. 112]

¹⁰While Backus claimed he knew little about the ENIAC [369], he drew flow diagrams in 1951-2 which quite closely resemble Goldstine & von Neumann’s notation [30].

$$\begin{array}{l}
 2 \quad \text{SUMA} = \text{SUMA} + C(I, J) * P ** (I - J) \\
 \quad \quad \quad \cdot Q ** J \\
 \\
 1 \quad \text{SUMB} = \text{SUMB} + D(I) \cdot P \cdot Q^{K+I} \\
 \quad \quad \quad \cdot \text{SUMA}^{K-1} \\
 \\
 \sum_{i=1}^{K+1} D_i \cdot P^i \cdot Q^{K+1} \left(\sum_{j=1}^I C_{ij} \cdot P^{i-j} \cdot Q^j \right)^{K-1}
 \end{array}$$

Figure 5.4: Jottings from John Backus, written on a note while at the Arlington Hotel in Binghamton, NY, depicting translations between mathematics (bottom row) and what can be typed on an IBM keypunch (top two rows), likely in preparation for a talk. Note the inconsistency in notation for definitions sumA and sumB: dots and exponentials could not be typed. *Photo of original* [31].

Unlike before, here ‘writing’ is no longer assumed the domain of the handwritten; instead, primacy is granted to the digitizing, standardized interfaces one grapples with, even when they are not there. Through this transition from writing to ‘writing’ (what can be typed), the serialization of notation enforced by typewriters and punch cards enabled an easy alliance with the metaphor of ‘language,’ a metaphor that even computing historians “forget... has its own history” [302]. In the Zuse and ENIAC visions, ‘language’ did not appear in any major way: Zuse preferred calculus and the term appears in GvN’s reports only once in reference to machine code [164]. By contrast, FORTRAN papers describe the notation as a language rather than as code or psuedo-code, announcing that this approach should “virtually eliminate coding” [37, p. 2]. Around this time, Grace Hopper and the popular media also played a large role in spreading the metaphor [302]. Typewritten visions of programming, rather than the Zuse and ENIAC visions, “[asked] what was possible to implement rather than what was

possible to write” [241, p. 15]. But the vision of typing code, despite its suggestion, still did not fully attach ‘writing’ to ‘typing.’ Following the division of labour in data processing at Remington Rand and IBM, among others, women were employed as keypunch typists on the UNIVAC (e.g., [352]) or in institutions that ran FORTRAN [243]. Statements were handwritten or typed onto paper slips called coding sheets and handed off to typists for punching [34]. It was not until punch cards were phased out in the electromechanical teleprinter era that typing “directly” into the machine displaced the need to handwrite code [143].

5.4 Discussion

These case studies of the origins of programming notation reveal several insights about the earliest history of HCI. First, ‘code,’ even in the current sense, was foremost handwritten and drawn before it was typed. The materiality of writing afforded alternative representations more closely associated with ‘notation’ than language. Second, the design of notations and practices for programming originally extended and adapted prior cultural activity. These adoptions included, but went beyond the simple use of natural language for keywords. Methods designed to suit one community’s culture, such as flow diagramming at the Moore School adapted from the practices of its electrical engineers, spread and were widely adopted with little reflection on their situatedness. Two of these methods, ENIAC and FORTRAN, thereafter delineated the dominant culture against which later approaches were measured and justified [396, 410]. Other methods, such as Zuse’s Plankalkül, remained ignored for cultural and historical reasons. Third and most importantly, programming notations and

their use are –and always were –social and material sites of intercultural conflict, compromise, and innovation. Especially for those marginalized from male, Anglocentric norms, programming did not just mean punching cards, flipping switches, planning or typing code, but was (and is) often “a problem of mapping from one culture to another” [109, p. 138], a recurrent site of translation work, cultural tensions and learning (see also [174, 14]). Beyond the translation work of social collaborations ‘around’ programming systems [318], however, I argue that translating between representations is a fundamental quality of the practice of writing code, and is not always the result of technical limitations.

Still, some readers may cling to the feeling that the shift-key keyboard was inevitable and typing code is actually the ‘best’ method (similar arguments to Brooks [65] and those who tried to explain the failure of Engelbart’s keyset [41, p. 217-8]). Indeed, like all standards in HCI [217], the metaphor of language and English keyboard interface did prove generative, such as inspiring approaches like Hopper’s COBOL [9] and enabling global traffic in code [174]. And it is also true that technical limitations hampered early alternatives such as GRAIL’s light pen coding [130], which Alan Kay called the most “intimate” interface he had ever experienced, but which suffered from a heavy stylus and low refresh rates [232]. But cases like Zuse’s Plankalkül should give us pause, raising serious questions about how the generalizing of some early, highly situated designs. First, alternative approaches to programming are often framed and justified for publication as ‘educational’ (to inculcate newcomers into the old regime) or for mere ‘end-users’ –i.e., people who are not, in the end, ‘experts.’ The implication is, of course, that the typewritten is the domain of the expert, and the ‘visual’ (or anything else) is for “newbies” [404]. If we are to change, we must be willing to challenge the practices and values of experts (as, for instance, the

notation design work of Bob Coecke and collaborators have in quantum physics [91]). Second, and as I expand upon below, the way we speak about programming and measure programming knowledge centres the typewritten –from the ACM classifier of this paper (“History of Programming Languages”), to the organizational and theoretical focus on single languages, to gate-keeping exams like the AP Computer Science A [59], to the kinds of questions asked by HCI researchers and neuroscientists [326], whose experiments seek to influence pedagogy, evaluation, and design. For instance, in August 2019, one of the creators of a ‘language-independent’ coding assessment apologized for their claims of independence and claimed that “we as a research community haven’t thought deeply enough yet about the interaction between programming languages and cognition” [178]. Following situated theories of cognition, we must learn to see programming systems as cultural tools that are embedded in particular social activity. Keeping this broad point in mind, I now connect my work to other scholars and draw further insights this framing of HCI’s early history might provide to the current field.

5.4.1 Framing the Early History of HCI as Situated Knowledge and “Making Do”

On the one hand, many progenitors of writing code were in a relatively privileged position in their respective societies, with many having the time, education, and resources to invent a new practice, even if some were marginalized by their contemporaries. On the other hand –and unlike later ‘hackers’ motivated by revolution, liberation, or democratization [12] –many inventors were

motivated by the need to simplify the everyday difficulties of handling error-prone data processing machines, systems which had existed in a similar form for decades. These inventors were “making do” [14] with their particular situation and on-hand materials to make incremental improvements to their interactions with computers. Keyboard interfaces were appropriated not out of a suite of alternatives or by some leap of imagination, but because they were literally lying around, ready to be repurposed –just like the other written practices I mention.

This situated, contingent perspective on knowledge construction connects with standpoint theory and third paradigm HCI [193, 386, 123]. Drawing from Donna Haraway’s situated knowledges [192], standpoint theory argues that scientific or technological visions often present themselves as objective truth –in the parlance of programming, masking themselves as universal or general-purpose –but are in fact “coming from particular points of view and generated through particular mechanisms” [193]. For example, rather than seeing FORTRAN as the first ‘general-purpose,’ compiled notation, this perspective would argue that FORTRAN was specific to a domain, in exactly the same way as, say, Max/Msp [1] is a programming environment for musicians. Similarly, rather than seeing the rise and fall of flow diagrams as reflecting the failure of ostensibly ‘visual’ thinking [285, 315, 62], this perspective instead would argue that flow diagrams were ill-suited to the wide range of different contexts in which they travelled. It was not the ‘visual’ that was flawed –such a blanket statement reflects what Dourish & Mainwaring call a “colonial impulse” [123] –but the early computing culture’s habit of universalizing and marginalizing, coupled with the constraints of machines and infrastructure. This same habit of generalizing “[o]ne historical particularity... into a timeless and spaceless universality”

[417] drove the entrenchment of the imperative (i.e. FORTRAN) paradigm.

5.4.2 The Naturalization of the Textual/Visual Dichotomy

My analysis also builds on the broader historical marginalization of handwork as something outside of programming [358]. Zuse and ENIAC project members imagined coding as involving written and drawn forms, connecting to their dispositions and practices as artists, mathematicians, and engineers. Later, however, coding became imagined as foremost typewritten, and programming notations became described as ‘languages’ belonging to either ‘textual’ or ‘visual’ paradigms [62]. Zuse’s zig-zagged, row-crossing line challenges the very distinction between the ‘textual’ and ‘visual,’ revealing it as a fabrication tied to many factors: the early dominance of the keyboard, the metaphor of language, and the serialization enforced by early machines’ processes. For instance, a paper in 1995 recounts arguments against visual languages which claimed that they are “not equally acceptable for all” and rationalized the position with a myth about right-left brain hemispheres [315].¹¹ Over a decade earlier in a Turing Award talk, Kenneth Iverson justified a typewritten approach by claiming that mathematical notation “lacks universality” and the typewritten is instead “universal (general-purpose)” [216, p. 340] –indeed a stark value shift from early inventors’ deference to mathematical notation.¹² Ingold argues that this constructed boundary between the textual and the drawn “hinges upon

¹¹In cognition and neuroscience, a growing number of studies suggest that processing of ostensibly formal (symbolic) notations utilizes visuo-spatial, nonlinguistic parts of the brain [247, 11].

¹²Theorists might raise questions here about computational universality and Turing completeness. Although not my focus, I ask theorists to notice how notions of completeness are upheld through translation work, e.g. to Turing machines or lambda calculus, encodings for numbers, etc. One might also keep in mind that, as Felleisen notes, proving completeness does little for insightful PL design [139].

a dichotomy between technology and art that has become deeply entrenched within the modern constitution” but that “dates back no more than three hundred years” and has its genesis in the rise of industrial capitalism, division of labour, and routinization [214, p. 127].

To disrupt the textual/visual fabrication in future work, we might consider the translation work of even the most technical people. Two burgeoning fields with this property are machine learning and quantum computing. Quantum computing practitioners communicate via a variety of diagrams and notation, yet when ‘writing code’ for a quantum computer, APIs require users to translate these representations into a FORTRAN-like sequence of calls [27]. While this typewritten standard allows easy inter-operation with other code and infrastructure, it also perpetuates a value-laden idea that the ‘visual’ is, in the words of one user, for those in “kindergarten” [27, p. 7], echoing those who ridiculed Frege’s notation. Such statements, I argue, should not be taken at face value in deference to the ‘experts’ or user-centred design. Instead, we should pay attention to how our interfaces have naturalized and centred typewritten notations, precluding the possibility that alternatives offer improvements over the typewritten (e.g., diagrams for unruly tensor indices [91, p. 10]). I shall explore a quantum programming system that mixes the “textual” and the “visual,” the typed and the drawn, in Chapter 6.

5.4.3 Embracing Heterogeneity in Programming Practice

So far, I have mainly been concerned with questions of computing culture and history, rather than speaking more directly to the subfield of programming (usu-

ally appended with languages and abbrev. as PL). Today, many programming communities continue a tendency to be biased towards a single approach – towards a single language or a one-size-fits-all vision –rather than viewing programming as a practice involving interactions between a plurality of representations, practices, infrastructure, people, and (possibly contradictory) perspectives. This tendency is embedded in the way programming is taught, where content often focuses on learning a language or paradigm, and avoids other learning about how to contend with infrastructures setup to support programming, or communication between people, software, and indeed other notations [259, 318, 190]. Said another way, PL researchers’ continued attention to single languages, boxed-in categories, and traditional eschewing of HCI methods and factors [389] resists efforts to conceptualize and design programming systems as interminglings of practices and representations.

As I have suggested above, future work in programming system design can build on the lessons of the past by embracing, rather than avoiding, heterogeneity in programming practice (e.g., drawing diagrams in Jupyter notebooks that *are* the code). In part, such efforts may benefit from paying a deeper attention to the ‘translation work’ users perform when writing code and how new notations and practices extend existing culture (whether to support that existing culture, or to design new practices that reflectively reject it). Suchman’s concepts of “partial translations” and “artful integrations” are important resources here: that “in place of the vision of a single technology that subsumes all others (*the workstation, the ultimate multifunction machine*), [designers] assume the continued existence of hybrid systems composed of heterogeneous devices” [386, p. 99]. As Lindtner et al. argue, those that seek to alter the status quo might also draw from feminist concepts of “walking alongside” (roughly, toler-

ance and respect without comprehension) and “parasitic resistance” – “an entity that is dependent on a host yet pursues independent goals, including goals that go against the interests of the host” [262] –indeed familiar concepts to intercultural competence education [186], infrastructure studies of organizational change [381], and the tension of learning within a dominant culture articulated by Lisa Delpit in *Other Peoples’ Children* [116].

Finally, I return to the anecdote where we began: DynamicLand’s programming ecosystem and rhetoric of liberation. Although the keyboard and screen reform as an “obligatory passage point” [248], an alternative, optimistic reading considers the project as a parasitic resistance grafted onto the status quo that it will eventually consume. Indeed, multi-domain approaches to programming are increasingly gaining acceptance in coding communities, reflected in the confluence of paradigms supported by Python and JavaScript, in approaches like React and Darklang, and also in PL theory, expanding on an earlier body of work on foreign function interfaces [5, 275]. In particular, the metaphor of language is now extended to “multilingual,” [178] “polyglot” [190], or “multi-language worlds” [5]. While these approaches remain largely beholden to a typewritten, English status quo, they do represent a shift towards the embrace of the pidgin and creole, the hybrid, towards a kind of epistemological inclusivity. A prominent example is the typewritten, Lisp-based Racket, framed (perhaps strategically) as an educational language. Its manifesto declares: “A proper approach [to programming] uses the language of the domain to state the problem and articulate solution processes... [S]ystems will necessarily consist of interconnected components in several different languages” [140, p. 114]. Though Racket too tends towards a totalizing project (“there must be no need to step outside” [140]), its approach is rare in the field of programming languages

and represents a promising turn towards the intercultural.

5.5 Conclusion

“The future... is that which breaks absolutely with constituted normality and can only be proclaimed, presented, as a sort of monstrosity.”

–Jacques Derrida [117]

Is it possible that the phrase “to write code” will not immediately imply typing in the future? Although some in programming and HCI continue to centre typewritten approaches and deploy universalist language, a historical perspective suggests that notations and practices of programming are likely much more situated than we typically imagine. Those in HCI and CS education should keep in mind how programming notations are cultural tools that are products of intercultural tensions and compromise, and not neutral descriptors of algorithms or systems. In particular, work in programming and software engineering disciplines today often carry with them an epistemology constructed around the typewriter. With advancements in pen-based computing and machine learning, if handwritten coding becomes possible, this paper argues that we in HCI should not be so quick to reify notations developed under different material constraints (e.g., handwriting ALGOL [256]) or indeed to oppose an intermingling of approaches.

The development of new programming systems will involve not just appropriation of the past, but conscious reflection on –and sometimes rejection of –its practices, notation, and discourse, of how they have come to condition our bod-

ies and imaginations. The earliest history of electronic computer programming revealed disparagement of other ways of doing and being that diverted from Western European norms, such as sequential, left-to-right symbols, valuing abstraction (or obfuscation) over anything geometric. A close inspection of the record reveals that 'why' one notation or practice was chosen over another is more a cultural question than a purely technical one.

CHAPTER 6

DESTABILIZING CULTURE IN PROGRAMMING: THE CASE OF NOTATIONAL PROGRAMMING

“Language is... ‘an essentially heterogeneous reality.’ There is no mother tongue, only a power takeover by a dominant language within a political multiplicity. Language stabilizes around a parish, a bishopric, a capital... It evolves by subterranean stems and flows, along river valleys or train tracks; it spreads like a patch of oil.”

–Deleuze & Guattari [114, p. 7]

For the first half of this thesis I was concerned with social interactions; that is, interactions of humans around or through computer programming, particularly in educational settings. Yet, as the prior chapter explored, one cannot divorce the material from the social: they must be understood together, co-producing one another [219]. Seeing programming through a cultural lens must therefore inevitably bring into question the tools through which we program, the commonplace practices that we imagine when we hear the phrase “to write code” versus those which escape us –and why. Accordingly, while historical and STS analyses can be illuminating, we should not linger on analysis alone: we must build; we must prod; we must seek to change practices, not just discuss them.

As the final step of this dissertation, therefore, I developed a prototype system that attempts to disrupt the “textual/visual dichotomy” spoken about in the previous chapter. It bridges the “typewritten” and “handwritten” worlds through a new paradigm of programming that I call *notational programming*. Part of my endeavor is a kind of critical or speculative design: producing a prototype in order to probe peoples’ existing assumptions and values around “writing

code.” These cultural assumptions are broached precisely when they come into contact with a radically different infrastructure, one that assumes and supports freehand drawing as central to expert programming practice. The adoption of notational programming, therefore, does not just entail the development of new user interfaces or improved recognition of existing notations like flow charts, but an active reconfiguration of cultural practices, representations, and values that have historically arisen around programming.¹

To explore notational programming, I designed an extension to Jupyter notebooks, Notate, that provides the ability to open drawing canvases within lines of code, allowing functions to accept canvas objects natively as arguments. The architecture also passes these objects a reference to the local scope, enabling type-written variables to be referred to in the handwritten context and vice-versa. I call this interaction *implicit cross-context references*, extending prior work on bimodal programming by further blurring territories between ‘input’ and ‘output’ [196].

To test a notational programming interface and implicit cross-context references in a concrete domain, I chose quantum computing (QC). This choice was strategic: programmers for QC, even when typing code, regularly translate between circuit diagrams and text [167]. An exploratory paper by Ashktorab et al. noted the potential for pen-based computing in QC spaces [27], but no such systems exist so far. I developed a toy notation, Qaw, that augments quantum circuit notation with abstraction features, such as custom gate definition, bun-

¹The usability study portion of this work was conducted with the help of Anthony DeArmas, a student who took my HCI course in summer 2020. The section describing the general notion of notational programming was written with Michael Roberts and Shrutarshi Basu, who are PL theorists in, or have graduated from, the PhD program in Cornell CS. Tapan Parikh, my advisor, contributed throughout to editing the piece and posed suggestions for the study design. The majority of work in this chapter was published at ACM UIST 2022.

dled wires, and recursion. I implemented an interpreter for a subset of Qaw using deep learning and classical computer vision techniques. Note that the impetus for my system is not to push the boundaries of quantum programming or to claim the interface is “better” for quantum programmers, but rather to explore the design space of notational programming in a very concrete domain.

To explore the efficacy of the Notate and Qaw prototypes, I then ran a study (with the help of a senior undergraduate student in CS, Anthony DeArmas) with 12 programmers who were familiar with Python and computational notebooks but novices to quantum programming. Participants were given six circuits of increasing complexity and tasked with programming them into the machine. Results show that almost all participants found the concept of implicit cross-context references intuitive; however, feedback suggests further improvements can be made to debugging infrastructure, interface design, and recognition rates. To validate the approach, I also compared Notate and Qaw to a typical typewritten workflow for quantum programming using the IBM Qiskit API. Results show that, for Python programmers, Qaw was comparable to Qiskit in terms of performance time, but suggest that further research is needed to understand the relative advantages of each approach.

To the best of my knowledge, this system is the first to explore a handwritten, diagrammatic paradigm for quantum computer programming (following the suggestion of sketch-based interfaces made in Ashktorab et al. [27]). The rest of this chapter is organized as follows: the front half covers related work (Section 2), a general description of a notational programming system (Section 3), and a case study with designing Qaw notation for quantum circuits (Section 4). The back half covers the evaluation of Notate and a subset of Qaw: usability

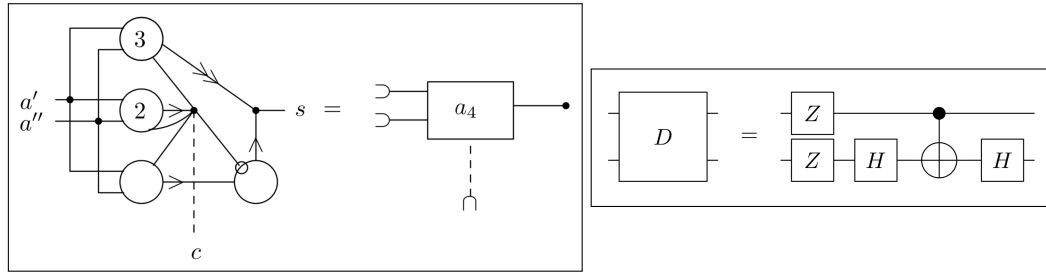


Figure 6.1: Circuit equalities from the advent of computing to quantum computing. Left: A circuit equality written by von Neumann in *First Draft of a Report on the EDVAC*, 1945 [415]. Right: a circuit equality from a quantum computing text.

study design (Section 5), findings (6), and comparison with a typewritten API (7). Finally, the discussion (8) serves to reflect on my design process, rationale, and comparison with graphical user interfaces (GUIs).

6.1 Related Work

In this section I summarize some prior work in programming and HCI and drawing-based interfaces related to this work. Interfaces for quantum computer programming will be covered in Section 6.4.

6.1.1 Programming Systems and HCI

A rich tradition in HCI focuses on developing novel interfaces for programming. One focus has been on systems for educational or novice users to make entry-level CS more accessible. These include block-based GUI environments, tangible programming with physical objects, or manipulatives in virtual reality (e.g., [425, 423, 421, 444]). More recently, a growing community of re-

searchers explore intersections between the fields of computer programming and human-computer interaction, or PL+HCI. Some examples of such work include enabling computers to complete unfinished programs [173, 308], constructively critiquing the design of popular PLs [443, 90], adapting usability methods for introducing new features to existing languages [89], understanding task-switching between languages [191], and using machine learning or crowd-sourcing to support code generation [408, 368, 289].

Some programming environments seek to explicitly or implicitly blend “visual” and “textual” representations. Max/Msp and various game engines such as Unity and Godot, for instance, foreground flow diagrams as their main programming interface but retain the ability to customize blocks with textual (typewritten) code in languages such as JavaScript and Lua. Rarely, however, are visuals interspersed within typewritten code [62]. One contemporary exception is the computational notebook paradigm popularized by the iPython notebook platform, which has received major attention by HCI researchers [360, 83, 195, 419, 226, 436, 420, 235]. Work in this area includes “bidirectional” coding, where “visual” and “textual” modalities are mixed in the form of click-and-drag GUIs, and edits to one affect the other [196, 436]. Other work explores the integration of GUIs inside computational notebooks for visualizing and navigating data [235, 307, 15]. An early precursor to this type of work is the “heterogenous visual languages” vision of Erwig & Meyer [136], which is perhaps closest to the vision I will lay out here, albeit without the focus on pen input.

6.1.2 Pen-based interfaces for programming

Pen-based computing has a long history of intersection with programming interfaces, starting from Iverson's SketchPad and continuing to sketch recognition of diagrams and pen gestures [17, 246, 111, 390, 130]. Some relevant recent examples are the preliminary demo of Inkbase (2021, SPLASH), the extensive notational programming work of Saquib (2020) in mathematics education, Microsoft's Sketch2Code, and AirBnB's sketch interface [x,x] (as well as my own undergraduate thesis on "notes with function"; unpublished manuscript, 2013). The dream of pen-based programming environments extend at least as far back to Rand Corp.'s GRAIL interface, which made the flow-diagram coding of von Neumann et al. interactive and inspired elements of Alan Kay's FLEX system [232]. Within HCI, sketching interfaces have been applied to support UI designers, such as in James Landay & coauthors' SILK and DENIM systems in the mid-90s to early 00s [245, 246, 260], as well as work by Ellen Yi-Luen Do, Mark Gross, Tracy Hammond and Randall Davis [172, 187, 221, 111]. Some of this work has sought to make sketching more interactive, offering tight sketch-interpretation feedback loops where shape gestures are successively recognized and/or beautified [130, 370]. Other systems convert handwritten diagrams into code within unidirectional workflows from early-stage sketches to textual code [282, 387]. One such example is Li et al.'s AlgoSketch, which supported recognition of code-like lines of freehand mathematics notation [256]. But while these and other systems have converted handwritten notation into computer programs [256, 124], to the best of my knowledge, no drawing interface has been embedded within a typewritten programming environment, while allowing for implicit communication between handwritten notation and textual code.

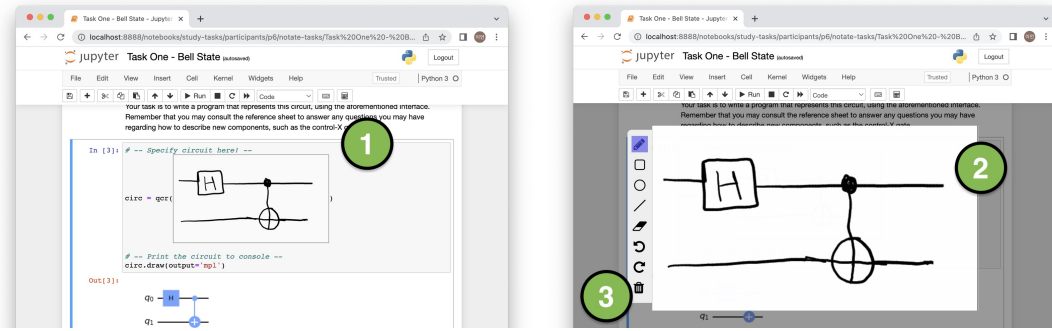


Figure 6.2: The main interface to the Notate system, embedded in a Jupyter notebook: (1) a canvas torn open inside a line of code in a cell; (2) fullscreen mode, accessed by touching or clicking on the canvas, with (3) a rudimentary toolbar.

6.2 What is Notational Programming?

Here I define the key features, principles, and rationale behind the notational programming paradigm. The Notate interface, depicting drawing canvases inside typewritten code, is shown in Figure 6.2. Users may draw on the canvas (1), resize it by dragging the corner, or click/touch it to open up fullscreen view (2), which presents a rudimentary toolbar (3). Canvases in a notebook cell move in response to text edits in the editor (e.g., newlines) and can be deleted, copied, and pasted alongside textual code, analogous to the “interactive visual syntax” GUIs of Andersen et al. [15]. The interface also allows users to paste in images from outside the notebook to instantiate a new canvas. Figure 6.3 depicts an example of copying a quantum circuit from Google Image search that is interpreted into an IBM Qiskit QuantumCircuit object.

To illustrate a simple workflow, consider the “notational program” in Figure 6.4. Here, a user has specified two vectors b and c as 2-tuples in Python code. Below these declarations, the user has drawn a diagram of the kind found in an introductory geometry class, depicting these vectors, writing an angle

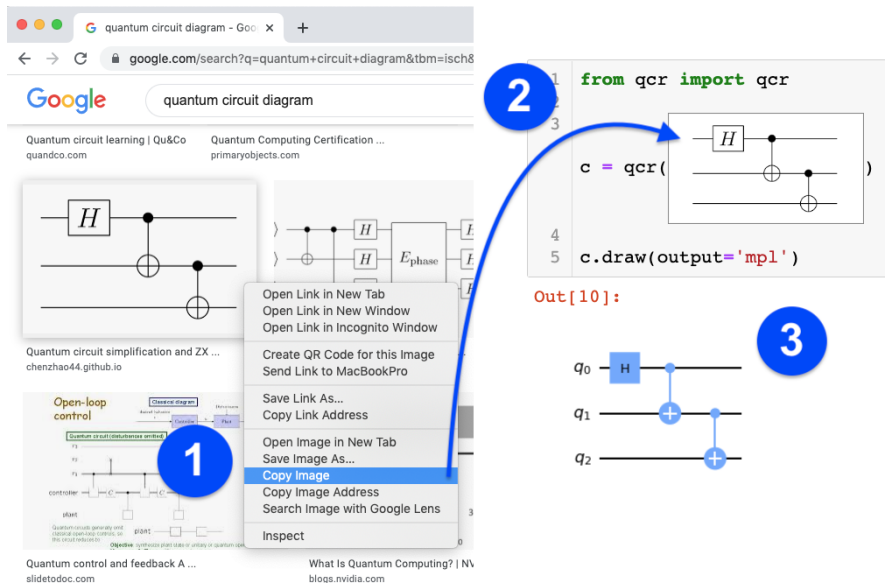


Figure 6.3: A user copies an image of a quantum circuit from search results (1) and pastes it directly into a function call (2). The user runs the cell and views output (3), verifying that the interpretation is correct.

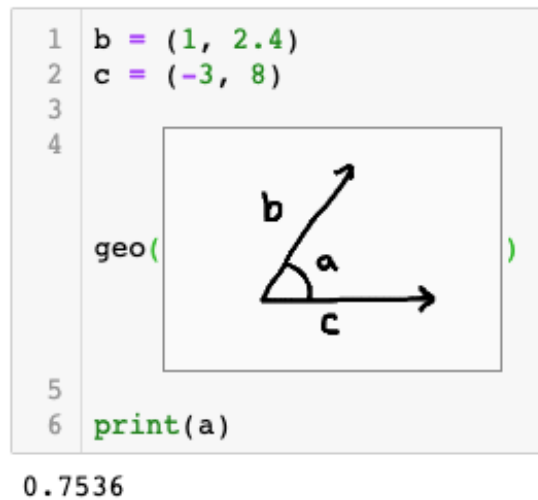


Figure 6.4: In a code cell, a user draws a diagram to calculate the value of angle a between two 2d vectors, b and c , defined as tuples in Python code. The Interpreter `geo` takes a Canvas and (implicitly) a reference to the local scope. Interpreting the diagram, it associates label a with an angle, realizes that a is not set, and declares it as a new variable in the host scope.

symbol between them, and labelling the angle α . To declare how their notation should be interpreted, they wrap their drawing canvas in a call to `geo()`. The `geo` interpreter then segments and recognizes portions of the drawing, matches labeled parts of the diagrams to information already in the current Python context, solves for the values of undefined variables (here, α), and binds them in the host scope.

This example illustrates what I call *implicit cross-context references*: the typewritten b and c becomes `b` and `c`, resp., while the handwritten α implicitly declares a Python variable `a` in the typewritten context. Note that the meaning of a diagram need not be literal: c in fact points in a different direction than what is drawn.

6.2.1 Definition and principles

Now that we have built some intuition, I provide a general definition. A notational programming system consists of three components: a *host environment*, a *pen-based interface*, and a *communication protocol* by which they interact. By host environment, I refer to a typewritten or drag-n-drop IDE with a corresponding “host language.” Here, the Jupyter notebook interface and Python are the host environment. By *notational programming interface*, I mean a system where:

1. users can **interact** with drawing canvases as first-class pieces of “code” embedded inside the host environment (copy and paste, drop in, delete, etc.)
2. users can **draw on** or otherwise edit the canvases using drawing features, with a stylus, touch, mouse, etc.

3. the system facilitates a **communication protocol** between the typewritten and handwritten contexts

A communication protocol specifies how the host environment “sees” image canvases embedded inside of it: what underlying types or objects they represent. In my implementation, the Python environment reads the canvas as an image object (a NumPy array) extended with some additional metadata, such as strokes, pressure data and timestamps. Importantly, the metadata includes a snapshot of the *local scope* captured at the point of execution (when the line of code with the canvas is read by the Python kernel). For brevity, I shall call this image-plus-metadata a Canvas object.

At the most basic level, one could use the notational programming system to set a variable directly equal to an image, or otherwise use it in a function call, without having to first save image data to a file.² The Canvas can, moreover, be something users create or edit using standard drawing tools. The intention of a notational programming system, however, is not simply the ease of importing images, but on handwritten notation as a *first-class element* when defining computation. To allow this, a notational programming system facilitates cross-context communication between labels in a handwritten *notation* (a handwritten system of marks, signs, graphics, or characters with a syntax and semantics, such as math, music, state diagrams, etc) with typewritten labels (whether variable names, functions, classes, etc) in the host scope. By cross-context, I mean that not only can handwritten notation reference typewritten variables, but later typewritten notation can reference handwritten variables. I use “variables” broadly to mean any named object in the host scope: functions,

²This is similar to drag-n-drop functionality in Mathematica: <https://reference.wolfram.com/language/howto/GetAnImageIntoTheWolframSystem.html>


```

1
  query = geo(
2
3   c = query.ask(a=4, b=3)
4   print(c)
5

```

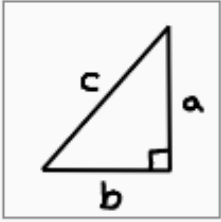


Figure 6.5: In a code cell, a user draws a diagram without prespecifying values for sides a , b and c . The diagram returns a queryable object. By setting some parameters (here, the `ask` method), the object returns the value of the remaining unspecified length. Note that this example is meant to express the possibilities afforded by notational programming, not argue the quality of this particular example’s API design.

classes, etc.

Practically, in order to interpret a notation in a particular execution context, one must write a notation Interpreter. This step is akin to defining a typed literal macro [307], albeit with a handwritten notation recognizer instead of a textual lexer. The process of interpretation can be enumerated into steps, roughly:

1. **recognition**: computer vision process visually recognizes the notation, syntactically; often this requires a segmentation step where symbols are extracted from “the rest” of the drawing and associated with parts of it
2. **semantic parser**: the recognized syntactic object is parsed in the notation’s semantics (potentially throwing errors or warnings, say for type mismatches or ambiguities)
3. **communication policy**: informally, a set of read/write rules between the host scope and the interpreter that specifies what variable names may be

“read” into the notational context, how they should be translated,³ and what typewritten variables may be declared or changed during the interpretation that are carried into the host scope. More formally, a read policy would specify both the domain of valid names and the expected types of the referenced variables.

Similar to macros for a typewritten notation [306], over the course of its execution, an Interpreter may:

1. Read certain variables (as in names) in the host scope
2. Modify existing variables in the host scope
3. Declare new variables in the host scope and bind them
4. Return a value (like a normal function)

An Interpreter that implicitly modifies or reads the local scope acts differently than a typical function, because (at least for Python) it may violate the scoping rules of the host language. For example, the normal Python code:

```
1 x = 0
2 def foo(img):
3     x = recognize_symbol(img)
4 foo(Image.load("handwritten_3.png"))
5 print(x)
```

outputs 0 to the console, because outer variable `x` cannot be set within `foo` (without the `global` keyword). However, the code:

³For instance, we may define a policy whereby Greek letters like θ declared in the notational context are accessible in later Python code by referencing the name `theta`.

```
x = 0
interpret( x = 3 )
print(x)
```

for some interpreter `interpret()` would print 3 (assuming, of course, the recognition step was successful). Here we violate Python’s scoping rules so that the Interpreter may act similarly to a line of typewritten code –which has access, implicitly, to variables defined in the local/host scope.

Finally, an object returned by an Interpreter may not be a direct value, but require certain parameters in order to specify its value (as in, a lambda function). Consider a triangle with labels a , b , and c that are not defined in the host environment (Figure 6.5). The interpreter `geo` returns an object “with holes” –that is, the meaning is indeterminate until it receives values for (some of) the undefined parameters, akin to a lambda function. For instance, one might use a method `obj.set(a=4)` to set a to length 4. Here, the object would return another object where $a=4$. This object would still need b or c to be defined, to infer the last side. The return object will typically need to do some unification and constraint solving in order to fill in these fields.

6.3 Case study: The Qaw Quantum Circuit Notation

Having described the notational programming paradigm in the previous section, I now narrow the scope to explore one potential application domain: quantum programming. This section reviews my motivations, design methodology, and specification for the Qaw (quantum-draw) notation and provides two ex-


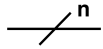

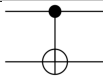
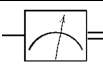
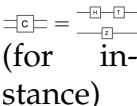
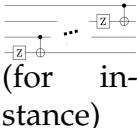
Notation	Term	Use	Representation in Qaw
	wire, qubit or "qubit line"	Stands for a single qubit (quantum-bit). Read from left to right.	(same)
	quantum register (or qubit bundle)	Stands for n qubit lines bundled together. Often used in more informal definitions of circuits.	$-/$ (without the n)
	gate	Stands for an operation (here, H) on the qubit(s) input into it (line(s) attached to left of box).	(same, but only capital letters accepted)
	controlled X-gate	Stands for a controlled-X operation on two qubits.	(same)
$ 0\rangle$	ket	Initializes the qubit line to its right (here, to state 0). Sometimes a complex state like $ \psi\rangle$ is written.	(same, but only 0, 1, +, or - currently supported)
	measure	Measures a qubit, collapsing the quantum state to a binary value.	$- $ (line with a stopper)
 (for instance)	assignment operation or subcircuit definition	Defines a subcircuit of the given name (here, C). The subcircuit can be used as a gate in later circuits.	With Notate, accomplished by setting a typewritten letter A-C equal to a drawing of the circuit wrapped in a <code>qcr()</code> call.
 (for instance)	ellipses operation (used informally)	Informal. The ellipses stands for repeating a pattern across n qubit lines. Pattern is inferred from surrounding context. Sometimes paired with parametrized gates.	Accomplished via recursive definition, which uses both assignment and slash-wire notation for bundling qubits. See Appendix A section A.0.8 for example.

Table 6.1: Some common elements of notation that practitioners use to write quantum circuits. For a full description of notation included in Qaw, see Appendix A. For an intro to quantum computing, see the Qiskit textbook [210].

amples. To aid readers less familiar with quantum computing (QC), Table 6.1 lists some common notation for quantum circuits, along with names, uses, and corresponding notation in Qaw. A full accounting of the notation included in Qaw appears in Appendix A. For an introduction to quantum computing, see the Qiskit textbook [210].

Quantum circuits are used to describe algorithms run on quantum computers. The general workflow of a researcher developing a new quantum algorithm is split into roughly two steps:

1. *Pencil and paper.* A programmer plans their algorithm by calculating in quantum mechanics and linear algebra notation, and draws a quantum circuit.
2. *Typing code and debugging.* The programmer translates their circuit into a typewritten programming language/API, outputs a diagram representation and inspects it, runs the code and observes the outcome, and edits and debugs.

Current software for quantum programming primarily supports step 2 of this process [27]. Typewritten approaches range from APIs in existing languages (e.g., Python for IBM Qiskit and Google Cirq [166, 210]) to entirely new languages (e.g., Microsoft Q# [391]). Researchers have also developed drag-n-drop graphical user interfaces, aiming to make quantum computing easier for novices (e.g., IBM Composer and Quirk [27, 149]). I was motivated to choose quantum computing for my case study after noticing how circuit diagrams remain a central feature of QC APIs and often appear side-to-side with typewritten code in many resources.⁴

⁴In Google Cirq, for instance, a circuit diagram is output as ASCII text to the terminal [166];

6.3.1 Notation Design Process

One of the first steps in designing notational programming interface involves designing a notation for a particular domain. To produce the Qaw notation, I conducted a survey of quantum programming resources. I surveyed circuits as they appear in papers on quantum algorithms, online tutorials and wikis, sketches in blog posts and class notes, example code, and textbooks. Given that one of the limitations of contemporary GUIs for QC is their lack of abstraction tools, I particularly paid attention to how authors handle abstraction in their circuit diagrams. I found several elements for denoting abstraction:

- Slashes bundle an abstract number of qubit lines, usually with a parameter n written above the slash
- Sub-circuits are declared using an assignment operator =
- Ellipses (...) are used to imply a repeating pattern within or across qubit and bit lines (e.g., last row of Table 6.1)
- Parameters appear either as arguments in parentheses or exponentials to a gate name
- A less common but powerful operation is recursive circuit definition, for instance in Rennela & Staton [353, p. 18]

I aimed to incorporate many of these prior conventions while designing the Qaw notation with an eye towards simplicity and reducing the effort required to hand-write elements. For instance, in Qaw one does not need to write a size parameter n for wire bundles above a slash, except in cases where one wishes

in IBM Qiskit, a diagram is drawn to a Jupyter plot [210]; Quipper outputs diagrams [167] and QuECT embeds ASCII quantum circuits into existing PLs [79].

to use the parameter elsewhere, needs to distinguish it from another, or wishes to implicitly define its size by inheriting from a typewritten variable. Nor do they need to explicitly write the tensor product in a gate (e.g., the \otimes^n in $H^{\otimes n}$), since the type of gate may be inferred from the type of the input. My choices were also governed by recognition accuracy –e.g., I opted not to include ellipses (as an option for suggesting repeated segments of circuits) since ellipses may be more prone to recognition error and ambiguity. Instead, the power of ellipses is obtained through recursive circuit definition.

While designing this notation, I paper prototyped how one would apply the notation to implement real quantum algorithms by handwriting solutions to tutorials in IBM Qiskit and Microsoft Q#, alongside the Python and Q# solutions to these tasks. I applied an iterative design process to amend the notation, reducing effort in favor of brevity where possible (e.g., the choice to depict the measure symbol as a capped output line, $-|$), or extending it (in the case of measures that then control later qubit gates, used in a quantum teleportation circuit). Nevertheless, just as programming languages like JavaScript are never “final,” so too do I expect notations to evolve and change as time goes on and more communities come into contact with the technology [225].

6.3.2 Examples: Superdense Coding & Grover’s Algorithm

To illustrate how Qaw works in practice, I wrote a small “notational program” for superdense coding (Figure 6.6), a common example quantum algorithm [325, 210]. Here, a user has specified bits a and b in typewritten code. They then drew a diagram that uses these variables to control gates X and Z , which

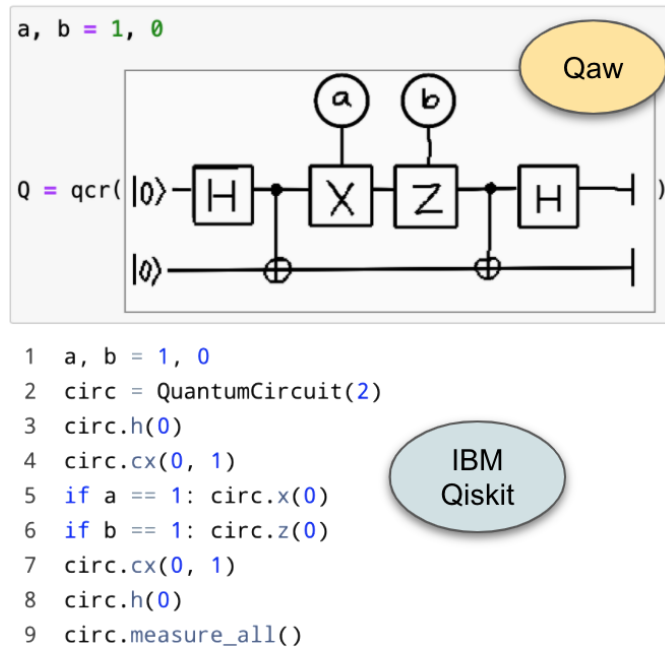


Figure 6.6: A common circuit for superdense coding, handwritten in Qaw (top) with Qiskit code for comparison. Notable features: 1) circles mark classical control bits a and b 2) kets initialize qubit lines, 3) stoppered outputs represent ‘measure’ operations. Here a, b variables are implicitly referenced in the handwritten context as classical bits that control gates.

essentially functions as an `if` statement –if a , then apply gate X ; otherwise, let the qubit pass through this part untouched. Measure notation –capped ends of output wires –indicate to measure both qubits. A later “run” method (not pictured) would then run the circuit Q on a quantum computer, observing the output. The equivalent Python code using Qiskit is depicted at the bottom of the figure.

For a more complex example, consider the general circuit for Grover’s Algorithm, as presented in the Qiskit textbook (Figure 6.7, top). This “abstract” circuit uses a common, albeit informal abstraction, the slashed-wire or “wire bundle” notation, to represent n inputs succinctly. The circuit may be coded in the current iteration of the Qaw notation using a similar slash (Fig 6.7, bottom),

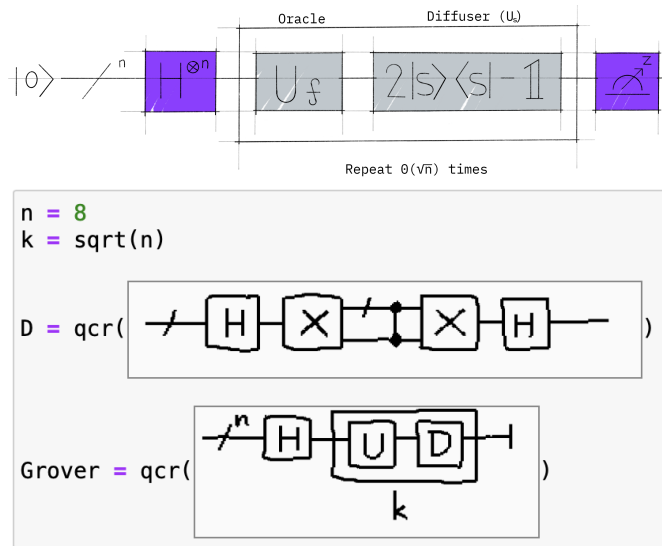


Figure 6.7: Top: a way to write Grover’s algorithm, from the Qiskit textbook, depicting the slash notation used in many quantum computing resources [210]. Bottom: the above circuit “coded” in Notate with a version of the Qaw notation, where \mathcal{D} is diffusion circuit and \mathcal{U} is the oracle (to be defined). The \mathcal{D} circuit is a solution to Task 3 in my user study.

here where the diffusion subcircuit D is also written using slash abstractions. To generate the same abstract circuit in Qiskit and Python requires using loops or recursion across n inputs.

6.3.3 Implementation

I implemented an interpreter `qcr` for a subset of the Qaw notation, using a combination of deep learning and classical sketch recognition techniques. The `qcr` function takes a `Canvas` and outputs either (1) an IBM Qiskit `QuantumCircuit` object or (b) an abstract wrapper over a Qiskit `QuantumCircuit`, called `AbstractQuantumCircuit`, which needs parameters (e.g., n for number of input wires) to generate a “concrete” `QuantumCircuit`. The abstraction is necessary as Qiskit does not support

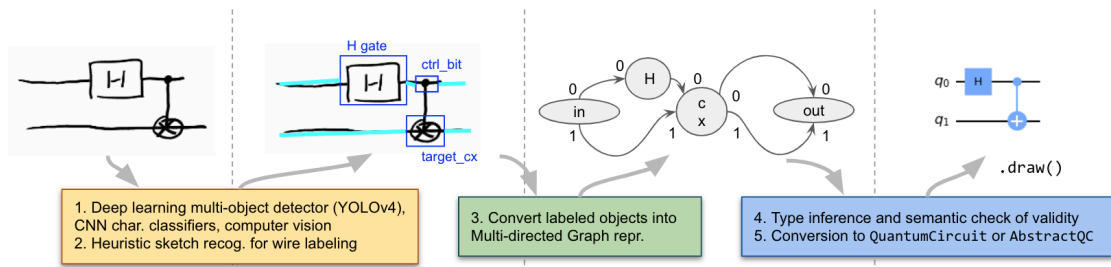


Figure 6.8: The process of interpreting a handwritten quantum circuit in my system. (*Drawing is P14's solution to Task 1.*)

specifying circuits with abstract elements. Including Notate, the full system took about a year for me to build, including (re)training of the ML model and an iterative process to improve the heuristic part of the algorithm. The architecture of the recognition system is illustrated in Figure 6.8.

6.4 Usability Study

Using this implementation, I proceeded to run a usability test of the Notate interface with a subset of the Qaw notation. My goal was to investigate how novices would use and perceive a notational programming interface in order to solve a series of QC tasks. I wondered especially about conceptual understandings of my core concept, issues around mode-switching between typing and drawing, and values participants might hold around different types of coding practices. Here I describe the study design, materials, and participants. I also validated the approach by comparing completion of the same tasks with a typewritten API, which I discuss in Section 7.

Task Design and Scaffolding

Since the Notate system targeted translations between diagrams and typewritten code, I settled on examining the “translation work” [21] users perform when programming quantum circuits into the machine. I designed a progression of tasks focused on a subset of the Qaw notation, designed to (1) introduce novices to quantum circuits and (2) focus on the implicit cross-context references concept, where custom gates, defined as typewritten Python variables with capital letters A-C, may be referred to in handwritten diagrams within the same scope. Much like a traditional API, the Qaw notation includes an array of complex components with nuances that could not be fully introduced within a 2 hour time-frame. Choosing to recognize only a subset also increases accuracy and reduces development costs.

I designed the tasks to lead up to asking participants to program a recursive circuit structurally similar to the Quantum Fourier Transform (QFT), omitting the parametrization and final multi-swap gate. Along the way, I chose some circuits specifically for their relationship to real quantum algorithms: Task 1 is a Bell State; while Task 3 is the n -ary diffusion subcircuit of Grover’s Algorithm. Other circuits were chosen for pragmatic or scaffolding reasons: Task 2 is a four-line circuit that I expected would require more time drawing than typing; Task 4 introduces the idea of defining subcircuits; and Task 5 introduces recursive definition. Task 6 tests all concepts that users had been taught across Tasks 1-5 (gates, controlled gates, slash-wires, subcircuits, and recursive definition). My designs of Task 5 and 6 were meant to provide harder programming problems than could be solved by the copying of an example diagram.

Participants

We recruited 12 participants (18-27 years old, median 20; 6 male; 6 female) who all self-reported prior experience in Python and using computational notebooks, but no prior knowledge of quantum computer programming.⁵ These participants had been randomly selected from a full pool of 24 participants (the remainder selected into the typewritten condition, discussed in Section 7). Given that quantum programming is rather niche, I anticipated that we could not recruit enough in-lab participants with prior expertise; even if we could, expert participants may be biased towards the quantum programming interface they are familiar with [89, 425]. Of those who participated, eleven were undergraduates, and one was a PhD student. Nine majored in CS, with others from Biology, Engineering, and Information Science fields.

Experiment Design and Procedure

After written consent, a member of the research team introduced participants to a Microsoft Surface tablet running the Jupyter notebook environment. All participants used the same Surface PC. Participants completed a tutorial, followed by six tasks of increasing complexity with an optional 5 min. break after the third task. Following the tasks, participants were asked to complete a Likert post-survey and a semi-structured interview. The post-survey asked for Likert ratings from 1 (strongly disagree) to 5 (strongly agree) for five questions, listed in Table 6.2. Each session was capped at 2 hours and participants were compensated \$30 in cash for their time.

⁵The screening criteria was: *“Participants must have prior experience in Python (taken a class, workshop, etc.), have at least cursory/passing knowledge of Jupyter notebooks, have no prior knowledge of quantum computing, and be comfortable drawing by hand.”*

Data collection, Setup and Materials

We asked to record the screen and microphone for the duration the study. A data logger captured user interactions with the Jupyter notebook, such as code cell edits, executions, tracebacks and toggling fullscreen mode. In addition, a researcher typed timestamped observational notes of the participant's interactions, with guidance especially to focus on anything not captured by the screen –e.g. shifting their posture, jotting on scratch paper, or moving the PC.

Participants were given a blank piece of scratch paper and a reference sheet. The sheet included circuit elements they would encounter during the tasks, and was made to mimic API documentation, since participants could not search online. Participants were told they may ask the researcher for a hint if they get stuck; and researchers were allowed to provide a hint if participants seemed to be stuck (e.g., repeating themselves due to confusions around an error). For the Notate condition, since my goal was not to test the accuracy of the recognizer, in the event of a recognition error on a correct (final) solution, the experimenter would let participants know their solution was correct and let them move on.

6.5 Findings

All Notate participants were able to complete the first five tasks, while nine were able to complete the last task within the allotted time (exceptions: P4, P6, P19). As shown in Table 6.2, post-survey results indicate non-normal distributions and high variances, indicating that there was a difference in how users adjusted to the interface. I affinity diagrammed the qualitative data and identi-

Q#	Question	Median	Mean	St dev.	Shapiro-Wilk (W, p)
Q1	The interface was easy to use.	4.0	3.58	1.1645	0.8596, p=0.0483**
Q2	The interface was enjoyable.	4.0	3.50	1.0000	0.8226, p=0.0171**
Q3	When I made a mistake/error, I found it easy to correct.	3.0	3.17	1.4668	0.8384, p=0.0265**
Q4	I felt confident using the interface.	4.0	3.75	1.1382	0.8512, p=0.0380**
Q5	When I completed the tasks, I felt like a programmer.	3.5	3.17	1.4035	0.9056, p=0.1874

Table 6.2: Post-survey Likert results for our Notate user study. Shapiro-Wilk tests indicate non-normal distributions at $p < 0.05$ for all questions except Q5; hence, I report medians. Variances are high, consistent with programming studies [89].

fied three major clusters of findings: conceptual understandings, error handling and debugging, and general usage patterns with the Surface device and Notate interface.

6.5.1 Conceptual (mis)understandings

Notate participant’s conceptual (mis)understandings could be broken down into three kinds: first, understandings about the core concept of implicit communication between typewritten and handwritten notations; second, understanding the Qaw notation and applying it successfully; and third, maintaining conceptual boundaries between what they considered “coding” versus “drawing.”

Understanding communication between typewritten and handwritten contexts

By far, participants had few, if any, difficulties understanding the core interaction of referencing typewritten variables within handwritten notation. They used it effectively to reference smaller circuits, typewritten as Python variables such as A , inside larger circuits, by handwriting \mathcal{A} (Figure 6.9). Notate participants would rarely bring up this interaction until pressed with a specific question by the interviewer. P19 recounts:

I didn't have any trouble with it. For the most part, it was able to recognize my handwriting and the variable that I typed.... was the same variable that I wrote down. [...] I guess it's the same intuition [as normal programming practice]. It's just instead of typing out a variable that you're referencing later, you're just writing it.

Three participants, to my surprise, even thought that in-line drawing canvases were an existing feature of Jupyter notebooks which they were simply unaware of (P4, P5, P11; e.g. “*Maybe this is like a feature in Jupyter notebooks?*”). These participants were otherwise familiar with Jupyter and had used it before. Said P5:

I wasn't sure if [the canvases] was a built-in library, or some library that already existed? Or if that's part of it? [Interviewer: Part of the Jupyter interface?] Yeah, that produces an image and then `qcr()` interprets the image.

A few participants described an initial hesitation followed by quickly becoming accustomed (P17: “*When I first read about it, it felt very foreign... [but] that*

idea seemed less foreign and sort of familiar once I did a couple of practices”). For two participants, one of the main conceptual difficulties stemmed from recursive definition. These users either did not notice the typewritten C (due to small font) in the Task 5 example diagram, or assumed the C gate had a different function. Said P11, who was observed scrolling quickly through the Task 5 text: “Maybe the C is just like, ‘lambda function abstracted away.’ [Like C] just means recursion. And I didn’t realize that the C was actually on the outside.” Examples of recursive definition are shown in Figure 6.9.

Understanding the Qaw notation and abstractions

Common conceptual errors of syntax across users are pictured in Figure 6.10. A few users also tried to slash multiple wires at a time, a feature that was unsupported by the implementation since it can lead to semantic ambiguities. Some of these confusions may merely be learning hiccups, but they may also suggest further extensions to the notation to support the varied ways people convey information (e.g., the third example in Fig. 6.10). When encountering these errors, the debugger would either print a warning, or raise an Exception with an error message. Common error messages included semantic “input / output size mismatch” errors when gates had different numbers of inputs than outputs (e.g., forgetting to draw the output wires to a gate). For Tasks 5 and 6, some participants also encountered “maximum recursion depth exceeded” errors that indicated their recursive circuit definition never terminated.

I anticipated that many participants would struggle on Tasks 5 and 6 due to the presence of recursion (a concept possibly rarely used in most participant’s programming in Python), requiring a complex understanding of Qaw abstrac-

tions. This, indeed, was what happened for *some* participants: for Task 5, both the median time *and the standard deviation* was approx. 12 minutes; for Task 6, median time was 27 minutes, st. dev 15.5 mins (for a full accounting, see Table 6.3). Closer examination of Task 5 reveals that two participants were able to complete the task in *a little over 1 minute*, and one about 1.5 min; while the longest two took about *30 min and longer*. Post-interview data indicates a possible reason for the faster participants: three reported that they felt comfortable with recursion and had taken (or were currently taking) a functional programming class. P21, for instance, a CS major, completed Task 5 in 5min 16s. When asked if anything in her background might have contributed towards her ability to solve the tasks, she said *“I understand recursion and know how to apply it.”* Her conceptual issues seemed less to do with recursion than with nuances of the notation, such as where slashes could go.

“This doesn’t feel like programming at all”: Negotiating boundaries between “coding” and “drawing”

“Coding begins with the drawing of the flow diagrams...”

–Goldstine & von Neumann, 1947 [164, p. 20]

One of my goals in conducting this work was to further explore how programmers negotiate boundaries between what practices constitute “programming” or “coding” and which do not, paralleling the historic separation between the “textual” and the “visual” and the dominance of the term “language” deriving from early adoption of the typewriter [21]. When presented with a series of tasks which effectively forefronts handwritten drawings in programming practice, I wondered whether it would destabilize or challenge participant’s no-

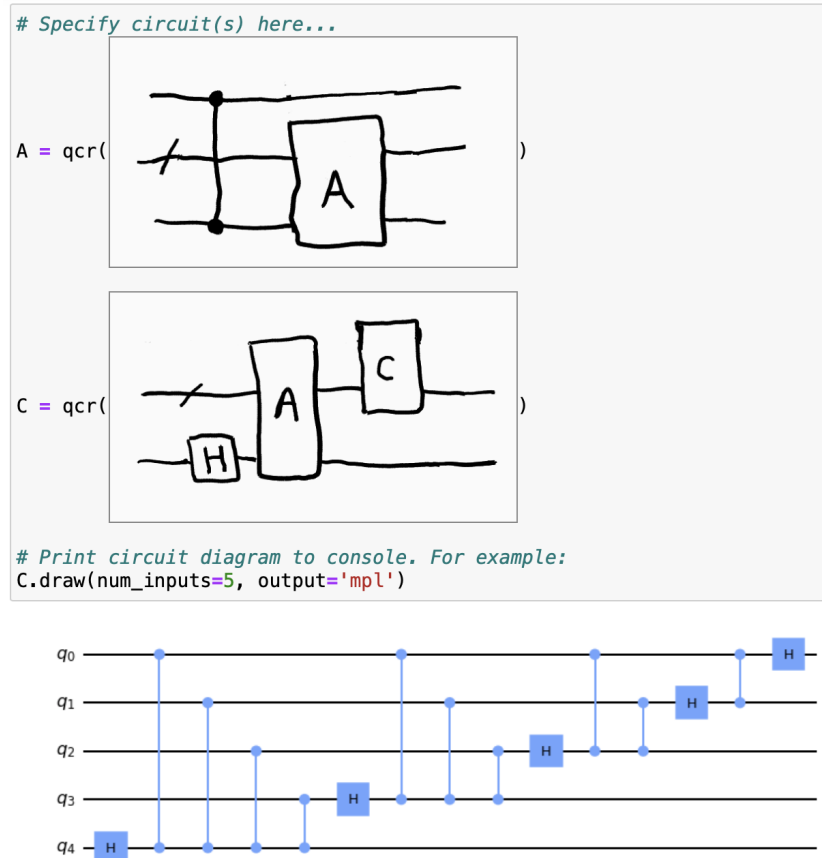


Figure 6.9: P5’s solution to Task 6, generating a pattern similar to the the body of the quantum fourier transform. Circuits \mathcal{A} and \mathcal{C} are defined recursively using Qaw slash notation and implicit cross-context references.

tions of programming.

Some Notate participants said that notational programming and especially recursive notation was unlike anything they had never encountered before (P11, a TA for an upper-level CS class: *“I don’t really think this compares to any other kind of programming that I’ve done in the past, like... It’s completely different”*) or struggled to compare it to their prior experiences. Participants would often erect boundaries between their experience using the drawing interface and what they considered “coding.” Said P21, when asked to compare her prior coding experience to the study:

This is definitely less coding; it was more drawing. For me coding is like, writing out... I guess, like this bottom line [points to C.draw()], like actually typing it up?

Participants appeared to associate the keyboard or “typing” with “coding” – often (somewhat ironically) resorting to the verb “writing” to describe how coding differed from “drawing” –in order to exclude drawing from the category of coding (P19: *“It [the study] is not what I was expecting... I thought I would have to write code”*; P7: *“When I’m programming, I don’t have to think about my handwriting... I just have to think about, you know, writing stuff”*). For them, “writing code” was a practice inexorably attached to the keyboard.

Yet, cracks in participants’ conceptual boundary-making could also occur. P7 began by claiming “this doesn’t feel like programming at all,” but as he tried to defend his point, ended up questioning how one would define programming, even asking the interviewer how they would define it. This provides evidence that, above and beyond technical concerns, notational programming interfaces may call into question participants’ ideas and values around what “programming” and “writing” code entails, especially as they sit with the concept for longer periods of time.

6.5.2 Error handling and debugging

In Notate, unlike a typical coding environment, errors may not be syntactic/semantic –errors caused by the user –but “recognition errors,” faults of the AI

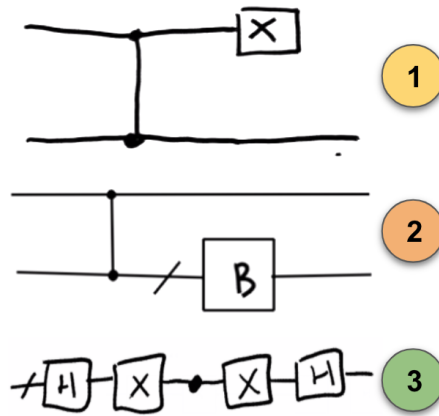


Figure 6.10: Three common “syntax errors” made by participants: (1) missing an output wire to a gate; (2) slashing an output wire of a control gate; (3) trying to use a single dot to represent a multi-Controlled Z gate in Task 3. In future iterations, we can imagine supporting some of these styles.

computer vision algorithm.⁶ Common to usability studies of notation recognition systems [304], during the studies we observed high variability in recognition rates for some Notate participants versus others. The post-survey responses provide some support for this claim: the question with the lowest score and highest variance is “When I made a mistake/error, I found it easy to correct” (Table 6.2). And indeed, those who struggled with the recognizer earlier or for longer durations may have rated the interface poorer. For instance, P14, who generally wrote in a very sketchy style, grew frustrated with the recognizer and gave the interface all 1’s in her post-survey, the lowest score across participants. By contrast, P21, who rarely encountered a recognition error, rated the interface all 4’s. In addition, comparisons with other pen-based interfaces may also be involved: for instance, one participant, in the post-interview, talked about their dislike of the Microsoft Surface, over the iPad & Apple Pencil interface that they were familiar with.

⁶Technically, a “recognition error” can also occur in typewritten coding if the syntax/grammar parser has a bug.

Conflations between semantic and recognition errors

In practice, the additional uncertainty of recognition errors meant that conceptual misunderstandings of the Qaw notation –where the user is learning, through trial and error, how to apply the syntax –were often confounded by recognition errors or issues parsing (or trusting) error messages.

If it had trouble, the recognition part of the interpreter would throw a plot of what the AI saw (Figure 6.11); but in practice, this could not catch errors when the AI *thought* it had parsed the drawing correctly, but the semantic parser then threw an error from the misrecognized circuit. This led to situations where participants interpreted recognition errors as semantic or syntax errors and vice-versa. For instance, while completing task 3, P7 struggled with the recognizer. They at first interpreted a recognition error as a semantic one, then tried a bunch more drawings before returning to a drawing semantically equivalent to their original solution –only this time, the recognizer worked, outputting the interpreted circuit:

Oh, come on, I did that the first time. [Frustrated, he looks at the output. It doesn't match the solution.] Is that circuit... Damnit. [pause] Oh, that's supposed to happen. Hey yo, no, I don't want to write that stuff [realizing he will have to draw a second line of gates].

Here, P7 is frustrated in realizing that what they thought was a semantic error was really the fault of the AI. This causes them to momentarily doubt the current, *correct* output –blaming the recognizer rather than their specification –before realizing that the AI was correct this time (“*oh, that's supposed to happen*”), and reattributing blame to themselves (“*no, I don't want to write that*”).

stuff"). Other participants would remark that a major similarity between coding in Python and in Notate was that error messages in Python were not always helpful. For instance, P3 mentions this similarity, along with the new source of "blame":

[In regular coding], the error you get is supposed to be helpful, and it's just not, or [it's] straight up wrong... But, being able to blame it on the image processing was kind of interesting. Like, "oh, it could be an error with me, or it could be an error with the computer."

Consistent with other work in HCI on improvisation and repair, sometimes these frictions were unexpectedly "productive" [227, p. 4]: for the wrong reasons, recognition breakdowns could end up nudging user's behavior in the right way. In other words, users could read *recognition* errors as a sign their circuit was *semantically* invalid –when it in fact was –and then stare at and amend their drawing towards the correct solution.

"It doesn't like my handwriting": Modifying drawings in response to errors

A common pattern, especially if recognition errors happened early on, involved participants amending their drawing practices to suit what they perceived the AI could understand, such as starting from freehand drawing to resorting to the rectangle and line tools. The sooner the recognizer failed, the faster and more extreme the amendments to their practices. P17, for instance, triggered a recognition error (with plot) on the Tutorial, the only participant to have done so. She then noticed the rectangle tool and began using rect and line tools for the rest the study. At one point she writes a fast Z, then erases it and rewrites it

Oops, I had a little trouble recognizing that. Here's what I saw:

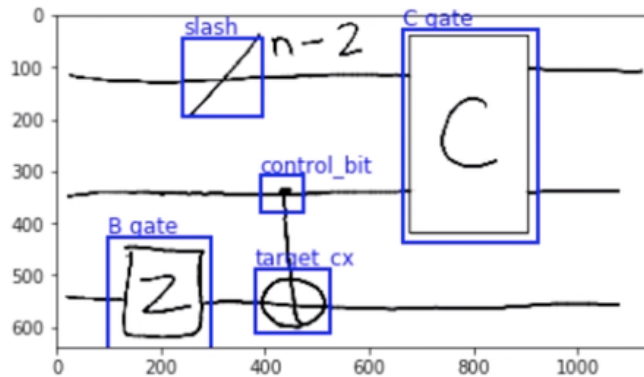


Figure 6.11: During Task 5, P8 encounters an error plot thrown by the AI recognizer (below the code cell). Noticing that a Z gate was mistaken for a B, P8 responds by erasing and rewriting her Z, then guessing the open corner was also an issue (“*Maybe it’s also this thing over here?*”) and filling it in. She runs the cell; it throws another error plot. The researcher present remarks on the $n-2$ as a conceptual misunderstanding –this notation was only used in a tutorial to explain how a recursive definition unrolls. P8 erases it and runs the cell, leading to the expected output (“*Alright! Looks good.*”).

slowly, possibly worried about the AI’s interpretation. Participants who rarely triggered a recognition error, by contrast, almost never used the rect, line, or circle tools in the toolbar.

6.5.3 Interactions with hardware and software

Many Notate participants would touch the screen while mode-switching between pen and keyboard. An especially popular mode of interaction was tapping in-line canvases to enter fullscreen mode, and tapping the background of fullscreen mode to close it. The Ctrl-\ shortcut to place a canvas at the mouse cursor seemed to be successful, with some participants only hesitating on Task 1 to remember the shortcut and the term `qcr()` (resolving the confusion by either using the reference sheet or looking back at the Tutorial). Strikingly, the

study did not encounter a case where participants, after Task 1, forgot to wrap their canvas in the `qcr()` function.⁷ One issue with the `Ctrl-\` shortcut was that, likely due to the weight and size of the felt Surface keyboard, some participants accidentally pressed `Ctrl+` (plus) when trying to use it, which enlarged the size of the browser content.

Many Notate participants preferred to use fullscreen drawing mode, with few writing directly on the in-line canvases. One exception was P8, who resized canvases to be almost the width of the code cell. So too would participants, especially for Task 5 and 6, use the scratch paper extensively, sometimes to write solutions pre-emptively on the paper before copying them onto a canvas. Their behaviors here may have to do with worries over the software interface, such as accidentally resizing the canvas, or hardware, such as the glass screen and how participants perceived the fidelity of the pen. Early in the study, P11 put the screen flat on the table, with the felt keyboard flat below it, but the keyboard got in the way, since it was near their elbow, leading to them returning the device to upright (stand) mode. Other participants would remark that they might have tried a “tablet” mode, e.g. P3: *“I would have been fine, every once in a while, when I had to type like `qcr()` just doing that on a touch screen... because the keyboard was almost entirely unused.”* Two participants asked for a “lasso” tool, referencing the Notability app they had used on their iPad. They wanted the lasso tool to move around parts of their drawings or copy parts of drawings within a canvas. Some participants mentioned having, or being familiar with, the iPad Pro and Apple Pencil; only one mentioned prior familiarity with a Microsoft Surface.

⁷In the Tutorial it had been explained that canvases are read internally as images, so possibly participants drew from their Python knowledge to understand `qcr` as a function taking an image argument.

Task	Notate (\tilde{x} , s)	Qiskit (\tilde{x} , s)	MWU (U1, p)	Levene w/ med (W, p)	Cliff's δ (effect size)
Tutorial	173.03s, 63.66s	150.19s, 87.66s	89.00, $p=0.3408$	0.75, $p=0.3957$	0.24 (small)
Task 1	59.27s, 25.71s	52.23s, 45.81s	82.00, $p=0.5834$	0.43, $p=0.5202$	0.14 (negligible)
Task 2	126.16s, 87.01s	77.81s, 62.48s	105.00, $p=0.0606^*$	0.51, $p=0.4837$	0.46 (medium)
Task 3	265.79s, 176.85s	422.07s, 464.89s	36.00, $p=0.0404^{**}$	0.80, $p=0.3813$	-0.50 (large)
Task 4	200.28s, 78.84s	233.71s, 284.03s	43.00, $p=0.0998^*$	2.58, $p=0.1225$	-0.40 (medium)
Task 5	721.54s, 722.15s	323.35s, 298.75s	83.00, $p=0.5444$	5.57, $p=0.0276^{**}$	0.15 (small)
Task 6	1619.33s, 925.31s	386.58s, 796.97s	125.00, $p=0.0024^{***}$	1.61, $p=0.2181$	0.74 (large)

Table 6.3: Task times (\tilde{x} =median, s =st. dev) and comparison statistics (Mann-Whitney U, Levene with median) across conditions.

(*= $p<0.1$ indicating a potential trend, **= $p<0.05$ indicating significant, ***= $p<0.01$ indicating highly significant)

6.6 Comparison with a Typewritten API

To assess the real-world performance of the Notate system, and to validate that the system wasn't significantly *worse* when compared to typewritten coding, I conducted a comparison of the interface with an alternate condition where participants had to use the IBM Qiskit API and Python to solve the same tasks. I chose Qiskit because it is one of the most popular quantum programming APIs. I also use this comparison to explore various tradeoffs between the two conditions regarding task types and workflows.

6.6.1 Participants

We recruited 12 additional participants (20-26 yrs age, median 21.5; 8 male, 3 female, 1 unspecified) who had been randomly selected from the full pool of 24 participants across studies, matching a between-group design.⁸ The study had the same timeframe of 2 hours and compensation of \$30, with Python-equivalent tasks and materials –a reference sheet to emulate API documentation, a piece of scratch paper, and a tutorial. All task explanatory text was altered to present the same conceptual information (as much as possible), except in typewritten code.⁹ Examples in tasks were presented as screenshots and could not be copied & pasted, to retain fairness across conditions. All Qiskit participants used the same Microsoft Surface as the Notate users except one, who used a MacBook Pro 2013. Participants could similarly ask for hints and, if they seemed stuck, the researcher present could provide help. I altered the name of the Qiskit API to ‘qcirc’ to eliminate potential biases around perceptions that a corporate entity had designed the interface.

6.6.2 Task by Task Performance

All Qiskit participants completed all tasks, although P13 requested significant help on Task 6 from the researcher present, resulting in pair programming the solution. Since a raw “time to task completion” metric does not factor in potential variability in reading the task text, which varies across conditions, here I estimate start time by the moment the user begins to work in the editor on the

⁸Eight were undergraduates, three PhD/masters students, and one unspecified; six majored in CS, the others in Engineering and Information Science fields.

⁹E.g., where Task 4 in the Notate study introduces subcircuits with an example of use, the Qiskit study presents example code to accomplish the exact same thing.

problem –editing the code cell to add a canvas or type code. End time is marked by the last execution of a code cell containing a correct solution, before starting on the next task.

Shapiro-Wilk tests of normality indicate that Qiskit completion time data is not normally distributed for all six tasks ($p < 0.01$ for every task except four, which is $p < 0.02$). For the Notate task times, none of the Shapiro-Wilk tests were significant, with only Task 2 near significance at $p = 0.059 < 0.1$. For post-survey subjective measures, a Shapiro-Wilk test reveals similar non-normality at $p < 0.05$ for all except one sample (Notate condition in Q5); and at $p < 0.01$ for all Qiskit post-survey questions. Because of these violations of normality and the between-subjects design, I report non-parametric tests to compare conditions: Mann-Whitney U to test for differences between distributions, Levine’s test with the median (a.k.a. the Brown-Forsythe test) for differences in variances, and Cliff’s δ for non-parametric effect size. I report median task times and other statistics in Table 6.3. Below, I unpack these findings with interpretations from qualitative data.

Tutorial and Task 1: No significant differences found. Medians and st devs. are close, suggesting that pen and keyboard input is comparable for entering a simple Bell State circuit.

Task 2: indicates a trend towards the Qiskit condition ($p = .06$), but does not reach significance. I included Task 2 to test the intuitive hypothesis that larger “concrete” circuits –that is, circuits without abstractions, with several gates and control lines that go in exact places –would be easier to solve via the keyboard.

Task 3: significant in favor of Notate $p = 0.04 < 0.05$, with a large effect size

(Cliff's $\delta=-0.5$). Task 3 introduces a new abstraction notation, the slash-wire, that participants have likely never seen before. The solution is also non-trivial (Fig. 6.7, subcircuit \mathbb{D}), and could have benefited from an extension to the notation whereby a dot captures a bundled multi-controlled Z gate (Fig. 6.10, #3). In both conditions, I did not include the multi-Controlled Z component in the task's tutorial explicitly, to emulate participants searching an API to solve the task. I did, however, introduce the slash-notation (in Notate) and (in Qiskit) remind participants of the "for i in range(n)" abstraction in Python for accomplishing similar looping over n inputs. The observations suggest that the multi-Controlled Z gate in Qiskit was a major pain point in the API, yielding confusion for participants both on where to apply it (trying to apply it within a for loop, initially) and how to enter qubit indices as an argument.

Task 4: suggests a slight trend in favor of Notate ($p=0.09$), but does not reach significance. Observations of the Qiskit condition indicate a potential pain point in the API regarding using custom subcircuits, with some participants appearing confused about the difference between output of `.to_instruction()` and subcircuit objects. In Qiskit, one had to cast their subcircuit to an instruction, then use the `.append` command with a list of indices as an argument; while in Notate, one could directly use their subcircuit by drawing a letter \mathcal{A} within a gate of their larger circuit.

Task 5: does not reach significance. However, Levene's test shows the two distributions have significantly different variances at $p<0.03$. In 6.1.2, I noted that the fastest three Notate users took under 1.5 minutes; the longest over 30 mins; here, the fastest Qiskit user took about 3.5 mins; while the longest took 18.5 min.

Task 6: significant at $p < 0.01$ in favor of Qiskit, with a large effect size (Cliff's $\delta = 0.71$).¹⁰ The relative speed of Qiskit users on Task 6 is as hypothesized by my choice in the task design. However, many Qiskit users were observed copying their solutions from Task 5 into Task 6, then amending inner calls, possibly accounting for the size of the difference.¹¹ In comparison, Notate participants, although they could copy canvases *within* a notebook, could not copy canvases from one notebook into another.

Likert Survey: Post-survey results indicate that Qiskit participants rated the interface as easier to use and more enjoyable compared to Notate ($p < 0.03$; median = 5 vs. 4 for both Q1 and Q2), while other metrics did not reach significance. For “When I made a mistake/error, I found it easy to correct,” a Levene's test with median yields a significant ($p = 0.018 < 0.05$) difference in variances, with higher variance for Notate. The high ratings for ease-of-use are somewhat unsurprising: after all, we had recruited participants experienced with Python and Jupyter notebooks, and the Qiskit condition did not ask them to do much more than use the interface they were already accustomed to. My choice of the word “interface” in the post-survey could also have led to confusion, as the post-interview data suggests that what some Qiskit participants were rating were Jupyter notebooks. Participants may also be inclined to blame themselves; for instance, P23 appeared to be struggling with Python and error messages, but in the post-interview did not attribute his frustrations to the interface, even when pressed.

¹⁰For the three Notate participants who could not finish Task 6, I include their times in the analysis without edit, for they took at least 10 minutes or longer.

¹¹P1, P23 and P24 copied the entire code cell; P15, P18 and P20 their entire recursive function; P9 a few lines; P10, P16 flipped rapidly between browser tabs to manually copy parts, including the base case; and P13 referenced it. Only P2 and P20 did not reference Task 5.

6.6.3 Qualitative observations

By contrast to Notate participants' reliance on touch and pen interactions and frequent use of the scratch pad on harder tasks, the Qiskit participants rarely used the scratch paper or touched the screen, and only one of them used the stylus to scroll with the scrollbar. Their means of interaction were the fold-out keyboard and trackpad of the Surface, akin to typical programming practice. Sometimes, a participant would be seen ready to write something on the scratch paper (pencil up), but then would return to the keyboard. A few participants used trackpad gestures to zoom out, such that they could see the entire notebook without scrolling. Across all participants, Qiskit users relied heavily on the provided example code snippets, sometimes copying example code almost verbatim before amending it for the task. As might be anticipated, the types of errors Qiskit participants encountered were also fundamentally different than Notate ones. Qiskit user errors centered around indexing and array-out-of-bounds errors (missing an index, going over by one, etc.). Users typically responded by revising the index argument, sometimes seemingly guessing in multiple quick edit-run-revise cycles until the cell ran without incident or gave the right output. By virtue of the notation, indexing errors were entirely absent from the Notate condition.¹²

Near the end of the post-interview, we asked Qiskit participants an interview question regarding a hypothetical drawing interface (they were not introduced to the Notate interface). Participants generally hypothesized that they would prefer to type code instead. When asked to expand, however, it materi-

¹²The idea that cumbersome indexing disappears in a diagrammatic notation aligns with the rationale of Penrose and Coecke & Kissinger, who designed index-free notations for tensors and quantum mechanics, respectively [327, 91].

alized that their belief stemmed from difficulties conceptualizing how drawing could define abstract circuits (i.e. over n inputs), with many assuming a tedious copying of the entire n -ary diagram in Task 6 (e.g., P1: “[Drawing] by hand?... I think that that would get out of hand so quickly”). And, when pressed about how to express abstraction visually, some Qiskit participants appeared at a loss for words. They could not imagine how that would be done. However, as a few participants continued talking, they appeared to grow uncertain of the solidness of their boundaries between programming and drawing. P10 for instance, without being aware of the Notate condition or ever seeing the interface, said:

[For] one of those abstract ones [circuits]... it'd have to be some like combination of drawing with like, notation, or something... But once you have that, then you're moving back into, like, the 'code territory'... If we were to do this completely, like in a 'no code' way, I'd probably have to completely draw [the circuit], right? But once you start thinking about ways to save time on that, like creating notation to define this abstractness, or the repeatedness... Technically, it's code, right?¹³

When drawings have notation, they “move into,” almost *invade*, the “territory” of code [114]. Since notational programming is, by design, meant to dissolve the “territories” of textual, keyboard-centric IDEs and handwritten drawings/notation, were such systems widely adopted, it is possible that the material assumptions underlying the concepts of “code” and “programming” would shift.

¹³During this study, the audio transcript reveals that the interviewer never mentioned the term “notation”: the participant came up with it on their own.

6.6.4 Limitations of evaluation

This study had numerous limitations. By far the biggest limitation is that Notate participants were novices, and could not conceptualize how such an interface might fit into a real quantum programming workflow. Future participatory design work with expert quantum programmers may enrich and amend this design. Second, my choice to recruit only those already experienced with Python and Jupyter notebooks may have biased them towards those familiar interfaces [89, 169]. Had I chosen participants with no knowledge of Python and Jupyter, it is an open question how the interfaces would compare.¹⁴ Third, and related to the first two limitations, it is unclear how results might change were users exposed to the interface for a longer duration of time (as the novelty of the interface and learning the notation make one-to-one comparisons with existing practice difficult). I also acknowledge that my comparison is limited to a typewritten API which might be improved in the future (say, to make using custom subcircuits more intuitive), and comparisons to other typewritten APIs may yield different results. Finally, because of the task design, Notate users did not encounter workflows that involved equal ratios of handwritten and typewritten coding. In practice, I envision more equal cooperation between input modalities. Future studies might explore tasks where participants learn both a typewritten API and a corresponding handwritten notation.

¹⁴There may also be no best design for a comparative study, however. According to Lieberman, comparative studies are only preferable when “changing the variable doesn’t change the paradigm of interaction... when the alternatives being tested are radically different from each other, you’ve got a problem” [257]. Greenberg & Buxton echoed Lieberman’s concerns, arguing that running comparative usability studies may (sometimes) be “harmful” for evaluating interfaces that challenge entrenched practices or norms [169]. They outline situations where either the interface is on the cusp of feasibility, but still too prone to errors; or where participants hold strong biases towards existing practices.

6.7 Discussion

Overall, my findings show that Notate users found the core interaction of implicit cross-context references intuitive. Moreover, all Notate users were able to apply Qaw abstractions to solve tasks 3-5, and most were able to solve the final task, involving double recursive definitions and at least one subcircuit, within the allotted time. In addition, although Notate participants were introduced to an entirely new notation, the slash-wire, in Task 3 –compared to Qiskit participants who used familiar `for` loops –they were able to complete the task significantly faster. This is all the more surprising since, usually, a portion of Notate participants’ task time was spent wrangling recognition errors. Comparison of task times for Notate vs. Qiskit conditions also provides evidence for a longstanding contention by programming researchers studying “visual” vs. “textual” notations: that which is “better” depends on the task at hand, how the design of the notation affords or resists encoding a particular solution, and the background and preferences of who is trying to apply it [168, 425]. Taken collectively, these findings support my “heterogenous” vision of notational programming –for designs that mix modalities, instead of demanding one for all time [386, 136]. I now reflect on future directions, rationale behind some design choices, and differences with GUIs.

6.7.1 Future directions and reflections on process

The qualitative findings for Notate revealed that, while participants could learn and apply Qaw abstractions, there was much room for improvement to the interface design. Recognition rates, drawing features (such as the lasso tool), task

reference material, and especially debugging feedback could all be improved. The infrastructure and standards around typewritten environments –linters, syntax highlighting, debugging feedback, etc. –has evolved over decades, and it stands to reason that handwritten programming could benefit from similar innovations. In Notate, only recognition errors ‘threw plots’ in their tracebacks, but UX-AI transparency principles [165] suggest that seemingly-semantic errors should also throw plots visualizing what the AI saw. Future work might explore best principles of building debugging toolchains for notational programming.

I also believe improvements can be made to my notation design process. Much of my work was akin to research-through-design, viewing the design artifact as an outcome “that can transform the world from its current state to a preferred state” [446, p. 493] –relating to how participants currently associate “coding” with the keyboard, versus (my intention) broadening this conception to include pen-based input. Here, the *process* of trying to design and implement a notational programming system may itself be a contribution, and offer suggestions for future practice. One improvement could be, before implementation, to run a Wizard of Oz (WoZ) study to examine how participants deploy a notation in practice, with the intent to fold their feedback into a more finalized specification. Another, more futuristic design is to leverage “evolving AI” [438] so that a distributed notational system would evolve its notation in response to how users actually use it, just as a non-computational notation like Feynman diagrams evolved as it came into contact with varied communities [225]. Relatedly, future work might center the communication protocol mediating the two contexts, and develop tools and guidelines for helping design domain-specific notations and interpreters.¹⁵ Past work on multi-domain sketch recognition by

¹⁵For instance, although I chose an “implicit” binding protocol here –where a subset of typewritten variable names are implicitly imported into the handwritten context and vice-versa –

Alvarado & Davis required a (typewritten) hierarchical shape description language to specify new domains [10]; here we might imagine combining deep learning-based recognition with programming-by-example techniques.

One additional question was raised by the “trick” many Qiskit participants used to solve Task 6 significantly faster than their Notate counterparts: the copying of one solution into a new notebook, followed by a short edit-run-revise cycle. Future designs might support copying Canvas objects not just across notebooks, but facilitate copying only parts of the drawing, or editing a finished drawing. Also, this raises a question of how one might design notations with localized copy-and-paste operations in mind.

6.7.2 Rationale behind turns of phrase

My design choice to integrate a drawing interface into typewritten IDE —rather than cordoning it off in a self-contained system —was inspired by my discussion of my historical analysis in Chapter 5, particularly Lucy Suchman’s guidelines to embrace heterogeneity: that “in place of the vision of a single technology that subsumes all others... (*the workstation, the ultimate multifunction machine*), [designers] assume the continued existence of hybrid systems composed of heterogeneous devices” [386, p. 99]. I worked to reopen a convention settled and stabilized earlier in history, recovering a heterogeneity lost from programming’s early moments of formation. In reality then, my proposed integration of typewritten and handwritten inputs was a *strategy*: a parasite that gloms onto the existing, dominant practice and seeks to corrode its constructed purity. “Cod-

perhaps some developers may favor more “explicit” bindings, involving passing more arguments to the interpreter.

ing” originally meant a handwritten, not a typewritten practice [21], and so it stands to reason that boundaries could again become blurred. To change the meaning of a term, however, requires shifting the material practices that underlie it.

The astute reader might wonder why I used terms like “notational programming system” throughout this piece, instead of more common phrases like “sketch-based interfaces.” Developers of sketching interfaces typically aim to support early-stage design processes and often take a user-centered design approach, aiming to recognize rather than reconfigure existing practice. For instance, Buxton defines sketching as any design process where the output is quick, plentiful, disposable, ambiguous, and with minimal detail [71]. Supporting sketching is an important area, as decades of rich research can attest. However, I preferred to use terms like “notational” and “pen-based” to, in part, clarify that I do not intend notational programming to support or augment early-stage sketching; i.e., it is not a replacement for paper and pen. Rather, notating is intended to be closer to the end product of user’s thought processes, more like typewritten code. Said differently, it is my intent that handwritten input requires a certain precision, just as typing on a keyboard does. That does not mean we should dismiss poor recognition rates, but rather –taking principles from AI research –calibrate user expectations appropriately [165].

Another key break with my approach was a focus on notation design and the ‘reconfiguration’ of user practices present in the PL community but often avoided in the framing of sketch recognition research. Per its name, sketch recognition research tends to proceed from a user-centered design perspective that focuses on recognizing existing notations and design practices (e.g.,

molecular diagrams, flow charts, or UI sketches), rather than taking a cultural-historical approach which conceives of writing practices as produced through a dialectic between human needs and material affordances [122, 21]. I perceive the goal of notational programming as not only recognition, but asking how we might *reconfigure* and extend existing notations –or indeed create new ones –in dialogue with new computational powers.

6.7.3 Handwriting interfaces vs. GUIs

A recurring question some people have asked when first hearing of this system is: “why not use a GUI?” Surely, they reason, an embedded GUI for the case of quantum circuits is preferable to a drawing interface –less prone to ambiguous errors, more immediate, editable, etc. And for certain purposes they may be right.¹⁶ My design was instead meant to be partly critical, i.e. “design that asks carefully crafted questions and makes us think” [125, p. 58]: I purposely avoided the tight feedback loops present in on-line sketch-based interfaces [371], seeking to challenge participants’ norms and values around programming (if a feedback loop isn’t required, then, technically, they could be writing significant parts of their program on a piece of paper). My goal was not universally faster or better products [125, p. 58], but to cause users to pause, to provoke or question their typical “ways of doing” programming.¹⁷ From the interviews, it seems that some participants’ understandings of coding were indeed broached or called into question after interacting with the system. Nevertheless, there may be practical benefits to my vision of the notational paradigm

¹⁶For instance, I would encourage IBM integrating Qaw abstractions like the slash-wire into their quantum circuit GUI Composer.

¹⁷Computer “coding” and “programming” are terms with a relatively recent history; their meanings have never been pregiven, fixed or static but have evolved over time.

that I now highlight: its shifting of front-end to back-end implementations, its altering of the “posture” of programmers, and its fluidity.

First, a single drawing interface (and protocol for reading its contents) means that implementers need not develop, embed, and test a new GUI widget for each new domain. Yes, new interpreters will need to be defined, but those methods need not unduly concern themselves with the particulars of the *front-end* interface. Although today it can be costly and time-consuming to iterate ML models [438], as these processes are streamlined, we can imagine future support tools that ease the process of developing said recognizers, say with transfer learning and few-shot examples.

Second, a GUI often –although not exclusively –assumes the familiar mouse/trackpad and keyboard setup. The way that laptops and desktop PCs constrain the body is often an assumed, and not reflected on, aspect of programming practice. Notational programming may change the paradigm of interaction towards pen-centric input, such that, hypothetically, one could be writing significant parts of a program on an e-Ink tablet. The body (posture, movements, even aspects of cognition) is constrained differently based on these setups, and some of the participants even reflected on this.

Third, and unlike domain-specific GUIs, because the core data for the notational paradigm are images, they can be copy and pasted and passed around at will, using existing, out-of-the-box infrastructure available on all operating systems. The image effectively is the program, or at least part of it. Were there an abundance of community-built interpreters, one could copy images from whiteboards, paper, or online resources, that then load them directly into data structures within a typewritten workflow. Like Mol’s characterization of the Zim-

babwe Bush Pump, we might say the notational programming interface aims to be a “fluid technology,” in that an image affords “a flexibility that allows it to travel almost anywhere” [112, p. 226]. What it sacrifices for a GUI’s “firmness,” in Mol’s terms, it may make up for in versatility and mobility.

6.8 Conclusion

“Wasp and orchid, as heterogeneous elements, form a rhizome... a capture of code... a becoming-wasp of the orchid and a becoming-orchid of the wasp. Each of these becomings brings about the deterritorialization of one term and the reterritorialization of the other...”

–Deleuze & Guattari [114, p. 10]

In this chapter, I introduced and evaluated a prototype notational programming system embedded in notebook environment. I introduced several principles of notational programming regarding how a host environment (here, type-written language and IDE) communicates with a pen-based interface for handwritten notation. As a case study, I then developed an abstract notation to writing quantum circuits, Qaw, and built a deep learning-powered interpreter of a subset of the Qaw notation. Interestingly, three Notate participants seemed to assume the in-line canvases shipped as a standard feature of Jupyter notebooks, and nearly all had no trouble grasping the concept of referencing type-written variables in handwritten code, suggesting that future users would find that core interaction intuitive. Such blending between the “textual” and the “visual” might also work towards shifting cultural values and boundary work around what “programming” entails. However, more work is needed on the in-

frastructure supporting notational programming –debugging tools, and designs that manage or mitigate the mode-switching between keyboard and pen.

Part III

Conclusion

CHAPTER 7

THE FUTURE OF PROGRAMMING AND HCI

Today, we stand on the precipice of widespread changes to programming practice. The advent of large language models (LLMs) in machine learning, trained on public codebases, has made the dream of “natural language programming,” which began with Grace Hopper’s COBOL, more feasible than ever before. Instead of ‘writing code’ directly, programmers can now prompt AIs by specifying their problem in a comment using natural language, and generate code that solves it. Advocates of this approach promise more efficient coding; at its most utopian, a radical alteration of programming practice. Yet as the DynamicLand example in Chapter 5 warns us, contradictions tend to shadow promises of radical change. From a cultural perspective, we see how LLM-powered systems rely on and reproduce existing languages, APIs, and workflows. They take as input existing ways of coding and reinscribe them, sometimes verbatim [249].¹ We face a situation where technological infrastructures, entwined with globalizing flows of capital, may be homogenizing cultural difference, coalescing towards global standards.² In such an environment, understanding programming as a cultural, rather than purely technical, practice has never been more important.

Throughout this thesis, I explored ways ‘programming’ and ‘culture’ in-

¹And with questionable legality. For instance, one Professor of CS showed that his LGPL-licensed code is reproduced almost exactly by GitHub CoPilot without a license: “@github copilot, with ‘public code’ blocked, emits large chunks of my copyrighted code, with no attribution, no LGPL license. For example, the simple prompt ‘sparse matrix transpose, cs_’ produces my cs_transpose in CSparse. My code on left, github on right. Not OK.” <https://twitter.com/DocSparse/status/1581461734665367554>

²Indeed, this can even be seen in the graphic design of logos for technology companies, coalescing towards bolder, sans-serif fonts that all look similar: <https://velvetshark.com/articles/why-do-brands-change-their-logos-and-look-like-everyone-else>

tertwine: the culture in programming, the programming in culture, and how phenomena under each banner may be challenged or changed with new activities, systems, and tools. Through my work in intercultural computing education, I explored how programming activities may serve as a site of intercultural learning, whether between students of different backgrounds, or as tools for reflection on cultural beliefs and assumptions ‘programmed’ into their society. Through my work on the culture in programming, I traced how the early history of ‘writing code’ enscribed such values and epistemological perspectives into programming practice, and suggested that this dominant culture has constrained the reception and/or creation of other values and practices. As a practical offshoot of my historical work, I then designed and studied one speculative, alternative vision of coding, *notational programming*, which imagines handwritten notation and drawing as central to expert practice.

Though I separated educational and material concerns into Parts I and II, respectively, the two are deeply connected. The material practices through which we ‘write code,’ and the dominant culture that has arisen around them, form the basis of, and determine the goals for, the education of newcomers. The goal of programming classrooms is to learn and repeat the representations and practices of the ‘experts,’ to come to value their ways of doing and seeing the world. Incentive structures are then set up to socially and economically reward those who best reproduce the existing culture.³ On the one hand, experts passing down methods to “organize and engage the world” [317, p. 2443] through pro-

³My experiences in academic programming language (PL) communities have reflected this. When networking in community retreats, the first question multiple people asked me was ‘who is your advisor?’ –and reacted negatively when they did not know the advisor, to the extent of quickly leaving the conversation. In other words, in PL communities, you are expected to serve as a conduit for your advisor, who ideally will be some kind of well-known figure (who were themselves advised by a well-known figure, etc). This came as a shock to someone who was used to HCI or information science conferences, where the focus on one’s advisor (implicitly, on reproducing an advisor’s views) was less stark.

programming is a positive endeavor, teaching the next generation how to access and change the powerful computing machines that fit in their backpacks and pockets. On the other, experts may reproduce values and practices in computing which constrain or devalue other methods or types of people [407]. In my U.S. study, the encounter between Evan and Jordan particularly underscores how the reproduction of the existing culture in CS intersects with the perpetuation of racial inequity (Section 3.3.4). Evan was not overtly negative or discriminatory –in fact he was polite and quiet –but rather seemed to regurgitate common messaging in CS about the value of demonstrating technical aptitude over any concern for social factors, at the expense of Jordan’s own enjoyment and care for her mother’s story. In the process of wanting to demonstrate his aptitude to a white male authority figure (the author or the teacher), Evan seemed to overlook the social goal of the activity, because he may have surmised from the wider society that social factors are ‘not relevant’ to succeeding in the field of computer science. As Turkle & Papert put it in 1990:

“Although the computer as an expressive medium supports epistemological pluralism, the [dominant] computer culture often does not. Our data points to discrimination in the computer culture that is determined not by rules that keep people out but by *ways of thinking that make them reluctant to join in*. Moreover, the existence of diverse styles of *expert* programming supports the idea that there can be different but equal voices even where the formal has traditionally appeared as almost definitionally supreme: in mathematics and the sciences.” [407, p. 132; *emph. added*]

While pedagogy and professionalization practices in computer science can

exhibit “ways of thinking that make [people] reluctant to join in,” aspects of these ways of thinking are actively encoded and maintained by material objects, such as programming environments and notations, that demarcate what practices are allowed and which require translation (“translation work”). Values that arise around dominant material practices like typing in C++ then serve to separate out the “real hackers” from the “rest.” Thus it is not surprising to find that students tend to express more *confidence* in their programming ability when typing in a textual editor, even when they are relatively more proficient in coding concepts when working in a graphical environment [254].

This association between (what constitutes) ‘real’ programming and typing was reflected by Notate participants, who tended to exclude drawing from programming practice, even though what they accomplished through handwriting was exactly the same as typing code, and even when it was, at times, faster to handwrite than type. Tellingly, when some participants (and later people reacting to the Notate work) expressed interest in the drawing interface, they often added that they were “visual people”; by contrast, when people expressed that typing code would be better, they could defer to a traditional belief that the keyboard and linear sequences of text are the ‘best’ way to program.⁴ As Lieberman put it, “People differ significantly in cognitive style that affects their reaction to interfaces. Nothing wrong with that, but it screws up experiments. Many interfaces, especially new and innovative ones, are controversial. Some people like them and some people hate them” [257]. The tendency in computer science to want to ‘optimize’ for the ‘best’ interface (embodied in my historical work by

⁴E.g., one researcher in PL theory expressed the belief that the idea that there are “visual people” is wrong and has been discounted by science. Since they themselves prefer to type code, therefore, it must be best for all. In fact, cognitive styles –or the idea that people have different aptitudes for different kinds of information –is a well-evidenced result in psychology [316], and research suggests correlations between differences in cognitive styles and cultural background [131].

inventors' early searches to find a 'universal language') is prone to prematurely devalue other methods of programming that privilege other types of people.

What might those charting the future of programming and HCI takeaway from this body of work? Overall, we might say that intercultural approaches to computing are focused on ontological change: rather than supporting existing practices myopically, without caring to question higher-level assumptions, one seeks to challenge existing practices and values and assumptions which undergird them. The designs which form the latter halves of Parts I and II –cultural algorithms pedagogy and notational programming, respectively –sought to disrupt existing assumptions and practices that divide and diminish the diversity of the world. These critical designs aimed to challenge how people saw themselves, others, and their societies. Going forward, then, those working at the intersection of programming and HCI *must be willing to move beyond user-centered approaches to design*, and (re)discover alternative philosophies that seek to challenge existing ways of being and doing.

One approach to help us reflect on (and challenge) the entrenchment of existing practices is the tradition of *ontological design*, developed by Winograd & Flores in the mid-1980s [433]. Its genesis as an inter-cultural endeavor, between scholars from the U.S. and Brazil, lends itself well to the “inter-cultural” sensibility of this thesis. And, its extension by Arturo Escobar incorporates Indigenous perspectives on being, knowing, and relationality which often challenge Western cultural assumptions and values [137]. Ontological design constitutes three beliefs:

“that design is something far more pervasive and profound than is generally recognised by designers, cultural theorists, philosophers

or lay persons; that designing is fundamental to being human –we design, that is to say, we deliberate, plan and scheme in ways which prefigure our actions and makings –and in turn *we are designed by our designing and by that which we have designed* (i.e., through our interactions with the structural and material specificities of our environments); that this adds up to a double movement –we design our world, while *our world acts back on us and designs us.*” [429, p. 70; emph. added]

Ontological design is therefore, for me, primarily a sensibility towards how *our world design us*, how history is, as Baldwin said, “[carried] within us”, how we “are unconsciously controlled by it in many ways,” and how “it is to history that we owe our frames of reference, our identities, and our aspirations” [38]. Ontological designing is designing to subvert current ways of doing and being – whether how programming education programs are constructed and conceived (as sites for learning programming, rather than building social relationships that challenge wider societal forces), or how “programming” is assumed to involve a familiar keyboard and mouse setup, and exclude (or devalue) visual notations and drawing. Like some Kenya-based students were open to change, ontological design asks us to learn from children’s perspectives, to take ways of thinking which may initially be seen as naïve or discouraged by dominant cultures, and build support structures which sustain them.

When we pursue ontological design, we open up the potential for change. The comments of some participants –one in the Kenyan study, and one in the notational programming study –point to the hopeful possibilities which can emerge: whether the Somali girl who initially hesitated to reciprocate with her

Sudanese partner, *“because he doesn’t understand that he’s a boy”*, claiming that she has *“a new friend [S25] who was my computermate and become my best friend”*; or the teaching assistant for a functional programming course who, after drawing abstract circuits, remarked that: *“I don’t really think this compares to any other kind of programming that I’ve done in the past, like... It’s completely different.”* From this perspective, AI and coding, as constituted by large language models taking the existing culture of code as input and reproducing it, does not fundamentally challenge the dominant culture. However, we see this not as the fault of “AI” technology, but rather how designers tend to frame the problem of applying machine learning technology to programming practice.

My hope for changing the culture in programming (and the programming in culture), however, is not without doubt. It could be that the existing, dominant culture of programming is merely entrenched in the future, and programming continues to be a relatively individualistic, siloed, typewritten activity that eschews care for relationships across difference or devalues other practices like drawing. Oftentimes, when a new approach is proposed that runs counter to the status quo, it is received less than favorably: whether upsetting existing dogma or power centers, perceived as less “technical” or scientific (leading to questions about why one would do such-and-such a thing, why we should care, or how we should quantitatively evaluate its effectiveness), or perceived as naïve (e.g., in the case of notational programming, the refrain “why not use GUIs?”). Rather than see such common reactions as a downside, however, I think such reactions can signal that we are on the right track, that we are posing important questions that may actually challenge existing practices; in short, that we are ‘ontological designing’ correctly.

In the future, I hope to expand upon topics and questions raised in this thesis. Pertaining to my work that mixes intercultural and computational learning, I believe I have barely scratched the surface in designing activities and pedagogy that investigates how computational thinking can serve the development of people's intercultural competence and societal understanding. Pertaining to my work that challenges tools and values around "writing" code, embedding a drawing canvases in a typewritten IDE opened up new questions such as: how might coding practice change when images are first-class objects in all major IDEs? How might pen-based tablets alongside AI technology open up new, interactive ways of programming that we can barely imagine today? And pertaining to both educational and material aspects of my thesis, can we construct a new way of programming that can bring people together but, unlike the DynamicLand project presented in Chapter 5, does not reproduce the dominant paradigm of typing code? How might these new programming systems evolve through community processes, including intercultural conflicts and encounters? What kinds of unique affordances might notational programming systems hold for intercultural learning in education programs, and vice-versa? While investigating such questions, I hope to maintain the curious, childlike science and technology studies' motto that "it could be otherwise," and go beyond critique to ask, "how exactly?": to build and design for the "otherwise," even when such designs may not be as warmly received or understood.

BIBLIOGRAPHY

- [1] Cycling '74. Max/msp/jitter, 2019.
- [2] Rediet Abebe, Solon Barocas, Jon Kleinberg, Karen Levy, Manish Raghavan, and David G Robinson. Roles for computing in social change. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 252–260, 2020.
- [3] Lila Abu-Lughod. Writing against culture. In *The cultural geography reader*, pages 62–71. Routledge, 2008.
- [4] Afrosocialists and Socialists of Color Caucus. Response to the Philly DSA, NYC Lower Manhattan branch and Adolph Reed: In Reed’s opinion, are we ever allowed to talk about race?, August 2020.
- [5] Amal Ahmed. Verified Compilers for a Multi-Language World. In *1st Summit on Advances in Programming Languages (SNAPL 2015)*, volume 32 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15–31, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [6] Sara Ahmed. Declarations of whiteness: The non-performativity of anti-racism. *Borderlands*, 3(2), 2004.
- [7] Atsushi Akera. *Calculating a natural world: scientists, engineers, and computers during the rise of US Cold War research*. MIT Press, 2008.
- [8] Ali Alkhatib, Michael S. Bernstein, and Margaret Levi. Examining crowd work and gig work through the historical lens of piecework. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, CHI '17*, page 4599–4616, New York, NY, USA, 2017. ACM.
- [9] Ben Allen. *Critical Approaches to the Materiality of Source Code: Between Text and Machine*. PhD thesis, Stanford University, 2017.
- [10] Christine Alvarado and Randall Davis. Sketchread: A multi-domain sketch recognition engine. In *ACM SIGGRAPH 2007 Courses, SIGGRAPH '07*, pages 34–es, New York, NY, USA, 2007. Association for Computing Machinery.

- [11] Marie Amalric and Stanislas Dehaene. Origins of the brain networks for advanced mathematics in expert mathematicians. *Proceedings of the National Academy of Sciences*, 113(18):4909–4917, 2016.
- [12] Morgan G. Ames. Hackers, computers, and cooperation: A critical history of logo and constructionist learning. *Proc. ACM Hum.-Comput. Interact.*, 2(CSCW), November 2018.
- [13] Morgan G. Ames and Jenna Burrell. ‘connected learning’ and the equity agenda: A microsociology of minecraft play. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing, CSCW ’17*, pages 446–457, New York, NY, USA, 2017. ACM.
- [14] Morgan G. Ames, Silvia Lindtner, Shaowen Bardzell, Jeffrey Bardzell, Lilly Nguyen, Syed Ishtiaque Ahmed, Nusrat Jahan, Steven J. Jackson, and Paul Dourish. Making or making do? challenging the mythologies of making and hacking. *Journal of Peer Production*, 12, 2018.
- [15] Leif Andersen, Michael Ballantyne, and Matthias Felleisen. Adding interactive visual syntax to textual code. *Proc. ACM Program. Lang.*, 4(OOPSLA), Nov 2020.
- [16] Subini Ancy Annamma, Darrell D Jackson, and Deb Morrison. Conceptualizing color-evasiveness: Using dis/ability critical race theory to expand a color-blind racial ideology in education and society. *Race Ethnicity and Education*, 20(2):147–162, 2017.
- [17] Georg Aritz and François Guimbretière. Crossy: A crossing-based drawing application. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology, UIST ’04*, pages 3–12, New York, NY, USA, 2004. Association for Computing Machinery.
- [18] Kwame Anthony Appiah. The conservation of "race". In *Black American Literature Forum*, volume 23, pages 37–60. JSTOR, 1989.
- [19] Kwame Anthony Appiah. *In my father’s house: Africa in the philosophy of culture*. OUP USA, 1993.
- [20] Kwame Anthony Appiah. *The lies that bind: Rethinking identity*. Profile Books, 2018.
- [21] Ian Arawjo. To write code: The cultural fabrication of programming no-

- tation and practice. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–15, New York, NY, USA, 2020. Association for Computing Machinery.
- [22] Ian Arawjo, Ariam Mogos, Steven J. Jackson, Tapan Parikh, and Kentaro Toyama. Computing education for intercultural learning: Lessons from the nairobi play project. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW):1–24, 2019.
- [23] Elizabeth Archuleta. Refiguring indian blood through poetry, photography, and performance art. *Studies in American Indian Literatures*, 17(4):1–26, 2005.
- [24] Debito Arudou. *Embedded racism: Japan’s visible minorities and racial discrimination*. Lexington Books, 2015.
- [25] Abena Ampofoa Asare. Exorcising “racecraft”: Toward the racesyllabus. *Radical Teacher*, 112:16–26, 2018.
- [26] Catherine Ashcraft, Elizabeth K. Eger, and Kimberly A. Scott. Becoming technosocial change agents: Intersectionality and culturally responsive pedagogies as vital resources for increasing girls’ participation in computing. *Anthropology & Education Quarterly*, 48(3):233–251, 2017.
- [27] Zahra Ashktorab, Justin D. Weisz, and Maryam Ashoori. Thinking too classically: Research topics in human-quantum computer interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI ’19, pages 1–12, New York, NY, USA, 2019. Association for Computing Machinery.
- [28] William Aspray. *Computing before computers*. Iowa State University Press, 1990.
- [29] Ian Ayres, Mahzarin Banaji, and Christine Jolls. Race effects on ebay. *The RAND Journal of Economics*, 46(4):891–917, 2015.
- [30] John W. Backus. Problem #29 flow chart for ibm ssec, 1951-2. John W. Backus Papers, Library of Congress Manuscripts Division, box OV1, item 71.
- [31] John W. Backus. Assorted notes on machine design, 1953-8. John W.

Backus Papers, Library of Congress Manuscripts Division box 5, item 79. (Written on the note: "Arlington Hotel in Binghamton").

- [32] John W. Backus. Can programming be liberated from the von neumann style?: a functional style and its algebra of programs. *Communications of the ACM*, 21(8):613–641, 1978.
- [33] John W. Backus. The history of FORTRAN I, II, and III. *ACM Sigplan Notices*, 13(8):165–180, 1978.
- [34] John W. Backus, R.J. Beeber, S. Best, R. Goldberg, H.L. Herrick, R.A. Hughes, L.B. Mitchell, R.A. Nelson, R. Nutt, D. Sayre, et al. The FORTRAN automatic coding system for the ibm 704 edpm: Programmer's reference manual. *Applied Science Division and Programming Research Department, International Business Machines Corporation*, 590, 1956.
- [35] John W. Backus, Robert J. Beeber, Sheldon Best, Richard Goldberg, L. Mitchell Haibt, Harlan L. Herrick, Robert A. Nelson, David Sayre, Peter B. Sheridan, H. Stern, et al. The FORTRAN automatic coding system. In *Papers presented at the February 26-28, 1957, Western Joint Computer Conference: Techniques for reliability*, pages 188–198. ACM, 1957.
- [36] John W. Backus and Harlan Herrick. Ibm 701 speedcoding and other automatic-programming systems. In *Symposium on Automatic Programming for Digital Computers, Washington, DC*, volume 13, pages 106–113, 1954.
- [37] John W. Backus, Harlan Herrick, and Irving Ziller. Preliminary report, specifications for the IBM Mathematical FORMula TRANslating System, FORTRAN. *Applied Science Division, IBM*, 1954.
- [38] James Baldwin. Unnameable objects, unspeakable crimes. *The White Problem in America, John Pub. Co. Inc., Chicago*, pages 170–180, 1966.
- [39] James Baldwin and Margaret Mead. *A rap on race*. Dell Publishing Company, 1972.
- [40] John Baldwin. Culture, prejudice, racism, and discrimination. In *Oxford Research Encyclopedia of Communication*. 2017.
- [41] Thierry Bardini. *Bootstrapping: Douglas Engelbart, coevolution, and the origins of personal computing*. Stanford University Press, 2000.

- [42] William Barnes, Myles Gartland, and Martin Stack. Old habits die hard: path dependency and behavioral lock-in. *Journal of Economic Issues*, 38(2):371–377, 2004.
- [43] Jean Jennings Bartik. *Pioneer programmer: Jean Jennings Bartik and the computer that changed the world*. Truman State University Press, 2013.
- [44] F. L. Bauer and H. Wössner. The “plankalkül” of konrad zuse: A forerunner of today’s programming languages. *Commun. ACM*, 15(7):678–685, July 1972.
- [45] Zvi Bekerman. Rethinking intergroup encounters: Rescuing praxis from theory, activity from education, and peace/co-existence from identity and culture. *Journal of Peace Education*, 4(1):21–37, 2007.
- [46] Zvi Bekerman. Identity versus peace: Identity wins. *Harvard Educational Review*, 79(1):74–83, 2009.
- [47] Zvi Bekerman and Michalinos Zembylas. *Teaching contested narratives: Identity, memory and reconciliation in peace education and beyond*. Cambridge University Press, 2011.
- [48] Ruha Benjamin. *Conjuring difference, concealing inequality: a brief tour of racecraft*, 2014.
- [49] Ruha Benjamin. *Race After Technology: Abolitionist Tools for the New Jim Code*. John Wiley & Sons, 2019.
- [50] Milton J. Bennett. A developmental approach to training for intercultural sensitivity. *International journal of intercultural relations*, 10(2):179–196, 1986.
- [51] Milton J. Bennett. Defining, measuring, and facilitating intercultural learning: a conceptual introduction to the intercultural education double supplement, 2009.
- [52] Milton J Bennett. Developmental model of intercultural sensitivity. *The international encyclopedia of intercultural communication*, pages 1–10, 2017.
- [53] Sebastian Benthall and Bruce D Haynes. Racial categories in machine learning. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 289–298, 2019.

- [54] Harold H. Beverage. Signaling, October 5 1937. US Patent 2,095,050 (Filed Apr. 26, 1933).
- [55] Katerina Bezrukova, Chester S Spell, Jamie L Perry, and Karen A Jehn. A meta-analytical integration of over 40 years of research on diversity training evaluation. *Psychological bulletin*, 142(11):1227, 2016.
- [56] Nicola J. Bidwell. Moving the centre to design social media in rural africa. *AI & Society*, 31(1):51–77, 2016.
- [57] Wiebe E. Bijker, Thomas Parke Hughes, and Trevor J. Pinch. *The social construction of technological systems: New directions in the sociology and history of technology*. MIT Press, 1989.
- [58] Su Lin Blodgett, Solon Barocas, Hal Daumé III, and Hanna Wallach. Language (technology) is power: A critical survey of "bias" in nlp. *arXiv preprint arXiv:2005.14050*, 2020.
- [59] College Board. AP Computer Science A, 2019.
- [60] Corrado Böhm. On encoding logical-mathematical formulas using the machine itself during program conception. translated 2016 by peter sestoft from corrado böhm: *Calculatrices digitales. du déchiffrement de formules logico-mathématiques par la machine même dans la conception du programme*. *ETH Zürich*, 1951. PhD dissertation. French original published in Bologna 1954.
- [61] Grady Booch. Oral history of John Backus, 2006. Computer History Museum.
- [62] Marat Boshernitsan and Michael Sean Downes. *Visual programming languages: A survey*. Citeseer, 2004.
- [63] Karen Brennan and Mitchel Resnick. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada*, volume 1, page 25, 2012.
- [64] Robert Brightman. Forget culture: Replacement, transcendence, relexification. *Cultural Anthropology*, 10(4):509–546, 1995.

- [65] Frederick P. Brooks. The mythical man-month: Essays on software engineering, anniversary edition. *Addison-Wesley*, 1975.
- [66] Simone Browne. Digital epidermalization: Race, identity and biometrics. *Critical Sociology*, 36(1):131–150, 2010.
- [67] Simone Browne. *Dark matters: On the surveillance of blackness*. Duke University Press, 2015.
- [68] Rogers Brubaker et al. *Ethnicity without groups*. Harvard University Press, 2004.
- [69] Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on fairness, accountability and transparency*, pages 77–91, 2018.
- [70] Arthur Walter Burks. Electronic computing circuits of the eniac. *Proceedings of the IRE*, 35(8):756–767, 1947.
- [71] Bill Buxton. *Sketching user experiences: getting the design right and the right design*. Morgan kaufmann, 2010.
- [72] Florian Cajori. *A History of Mathematical Notations (Vol. I and II)*. Chicago: Open Court Pub. Co., 1929.
- [73] Elizabeth H Campbell. Urban refugees in nairobi: Problems of protection, mechanisms of survival, and possibilities for integration. *Journal of refugee studies*, 19(3):396–413, 2006.
- [74] Hazel V Carby. The multicultural wars. *Radical History Review*, 1992(54):7–18, 1992.
- [75] Vauldi Carelse and Christian Parkinson. South africa’s coloured community: ‘still marginalised after apartheid’, 2020. BBC News Africa, Video.
- [76] Aaron Castelan Cargile and Howard Giles. Intercultural communication training: Review, critique, and a new theoretical framework. *Annals of the International Communication Association*, 19(1):385–404, 1996.
- [77] Prudence L Carter. *Stubborn roots: Race, culture, and inequality in US and South African schools*. Oxford University Press, 2012.

- [78] Stephen Cave and Kanta Dihal. The whiteness of AI. *Philosophy & Technology*, 33(4):685–703, 2020.
- [79] Arnab Chakraborty. Quect: a new quantum programming paradigm. *arXiv preprint arXiv:1104.0497*, 2011.
- [80] Anita Say Chan. *Networking peripheries: Technological futures and the myth of digital universalism*. MIT Press, 2014.
- [81] Daniel Chandler. The phenomenology of writing by hand. *Digital Creativity (Intelligent Tutoring Media)*, 3(2-3):65–74, 1992.
- [82] Kathy Charmaz. *Constructing grounded theory*. Sage, 2014.
- [83] Souti Chattopadhyay, Ishita Prasad, Austin Z. Henley, Anita Sarma, and Titus Barik. What’s wrong with computational notebooks? pain points, needs, and design opportunities. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–12, New York, NY, USA, 2020. Association for Computing Machinery.
- [84] Francesco Checchi, Adrienne Testa, Abdihamid Warsame, Le Quach, and Rachel Burns. Estimates of crisis-attributable mortality in south sudan, december 2013-april 2018. Technical report, London School of Hygiene and Tropical Medicine, 2018.
- [85] Wendy Hui Kyong Chun. *Programmed visions: Software and memory*. MIT Press, 2011.
- [86] Wendy Hui Kyong Chun. Race and/as technology, or how to do things to race. In *Race after the Internet*, pages 44–66. Routledge, 2013.
- [87] Paul Cobb. Where is the mind? constructivist and sociocultural perspectives on mathematical development. *Educational researcher*, 23(7):13–20, 1994.
- [88] Paul Cobb. Cultural tools and mathematical learning: A case study. *Journal for research in mathematics education*, 26(4):362–385, 1995.
- [89] Michael Coblenz, Gauri Kambhatla, Paulette Koronkevich, Jenna L Wise, Celeste Barnaby, Joshua Sunshine, Jonathan Aldrich, and Brad A Myers. Pliers: a process that integrates user-centered methods into program-

- ming language design. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 28(4):1–53, 2021.
- [90] Michael Coblenz, Michelle L. Mazurek, and Michael Hicks. Does the bronze garbage collector make rust easier to use? A controlled experiment. *CoRR*, abs/2110.01098, 2021.
- [91] Bob Coecke and Aleks Kissinger. *Picturing quantum processes*. Cambridge University Press, 2017.
- [92] I. Bernard Cohen, Robert V. Campbell, Gregory W. Welch, and Robert V.D. Campbell. *Makin'numbers: Howard Aiken and the computer*. MIT Press, 1999.
- [93] Michael Cole. Can cultural psychology help us think about diversity? *Mind, Culture, and Activity*, 5(4):291–304, 1998.
- [94] Michael Cole, Distributive Literacy Consortium, et al. *The fifth dimension: An after-school program built on diversity*. Russell Sage Foundation, 2006.
- [95] Mike Cole. Critical race theory comes to the UK: A marxist response. *Ethnicities*, 9(2):246–269, 2009.
- [96] Beth Coleman. Race as technology. *Camera Obscura: Feminism, Culture, and Media Studies*, 24(1 (70)):177–207, 2009.
- [97] Patricia Hill Collins. Black feminist thought in the matrix of domination. *Black feminist thought: Knowledge, consciousness, and the politics of empowerment*, 138(1990):221–238, 1990.
- [98] Patricia Hill Collins. Toward a new vision: Race, class, and gender as categories of analysis and connection. *Race, Sex & Class*, 1(1):25–45, 1993.
- [99] Patricia Hill Collins. *From Black power to hip hop: Racism, nationalism, and feminism*. Temple University Press, 2006.
- [100] Darlene Connoly. Beyond 'race relations'. *Jacobin Magazine*, January 2018.
- [101] James W. Cortada. *Before the computer: IBM, NCR, Burroughs, and Remington Rand and the industry they created, 1865-1956*. Princeton University Press, 2000.

- [102] Sasha Costanza-Chock. Design justice, ai, and escape from the matrix of domination. 2018.
- [103] Kimberle Crenshaw. Mapping the margins: Intersectionality, identity politics, and violence against women of color. *Stan. L. Rev.*, 43:1241, 1990.
- [104] Jeff Crisp. A state of insecurity: The political economy of violence in kenya's refugee camps. *African affairs*, 99(397):601–632, 2000.
- [105] Stanley Crouch. *The All-American Skin Game, or Decoy of Race: The Long and the Short of It, 1990-1994*. Vintage, 2010.
- [106] Emma Dabiri. *What White People Can Do Next: From Allyship to Coalition*. Penguin Books UK, 2021.
- [107] Jessie Daniels. Race and racism in internet studies: A review and critique. *New Media & Society*, 15(5):695–719, 2013.
- [108] Antonia Darder. What's so critical about critical race theory?: A conceptual interrogation with rodolfo torres. *Counterpoints*, 418:109–129, 2011.
- [109] Subrata Dasgupta. *It Began with Babbage: The Genesis of Computer Science*. Oxford University Press, 2014.
- [110] Paul A. David. Clio and the economics of qwerty. *The American economic review*, 75(2):332–337, 1985.
- [111] Randall Davis. Magic paper: Sketch-understanding research. *Computer*, 40(9):34–41, 2007.
- [112] Marianne De Laet and Annemarie Mol. The zimbabwe bush pump: Mechanics of a fluid technology. *Social studies of science*, 30(2):225–263, 2000.
- [113] Darla K. Deardorff. *The SAGE handbook of intercultural competence*. Sage, 2009.
- [114] Gilles Deleuze and Félix Guattari. A thousand plateaus, trans. brian masumi, 1987.
- [115] Richard Delgado and Jean Stefancic. *Critical race theory: An introduction*, volume 20. NYU press, 2017.

- [116] Lisa Delpit. *Other people's children: Cultural conflict in the classroom*. The New Press, 2006.
- [117] Jacques Derrida. *Of Grammatology*, trans. Gayatri Chakravorty Spivak, 1976.
- [118] Adrienne Dessel and Mary E. Rogge. Evaluation of intergroup dialogue: A review of the empirical literature. *Conflict Resolution Quarterly*, 26(2):199–238, 2008.
- [119] Robin DiAngelo. *White fragility: Why it's so hard for white people to talk about racism*. Beacon Press, 2018.
- [120] Frank Dobbin and Alexandra Kalev. Why diversity programs fail. *Harvard Business Review*, 94(7):14, 2016.
- [121] Paul Dourish. Not the internet, but this internet: how othernets illuminate our feudal internet. In *Proceedings of The Fifth Decennial Aarhus Conference on Critical Alternatives*, pages 157–168. Aarhus University Press, 2015.
- [122] Paul Dourish. *The stuff of bits: An essay on the materialities of information*. MIT Press, 2017.
- [123] Paul Dourish and Scott D. Mainwaring. Ubicomp's colonial impulse. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing, UbiComp '12*, page 133–142, New York, NY, USA, 2012. ACM.
- [124] Wenxiao Du. Code runner: Solution for recognition and execution of handwritten code. Technical report, Stanford University, 2012.
- [125] Anthony Dunne and Fiona Raby. *Design noir: The secret life of electronic objects*. Springer Science & Business Media, 2001.
- [126] Roger Echo-Hawk. *The magic children: racial identity at the end of the age of race*. Routledge, 2016.
- [127] Ron Eglash, Audrey Bennett, Casey O'donnell, Sybillyn Jennings, and Margaret Cintorino. Culturally situated design tools: Ethnocomputing from field site to classroom. *American anthropologist*, 108(2):347–362, 2006.
- [128] Ron Eglash, Michael Lachney, William Babbitt, Audrey Bennett, Martin Reinhardt, and James Davis. Decolonizing education with anishinaabe

arcs: generative stem as a path to indigenous futurity. *Educational Technology Research and Development*, 68(3):1569–1593, 2020.

- [129] Nadine Ehlers. *Racial imperatives: Discipline, performativity, and struggles against subjection*. Indiana University Press, 2012.
- [130] Thomas O. Ellis, John F. Heafner, and William L. Sibley. The grail project: An experiment in man-machine communications. Technical report, RAND Corp., 1969.
- [131] Jan B Engelmann and Marianna Pogosyan. Emotion perception across cultures: the role of cognitive mechanisms. *Frontiers in psychology*, 4:118, 2013.
- [132] Yrjö Engeström. *Learning by expanding*. Orienta-Konsultit, 1987.
- [133] Nathan Ensmenger. The multiple meanings of a flowchart. *Information & Culture*, 51(3):321–351, 2016.
- [134] Zimitri Erasmus. Contact theory: Too timid for “race” and racism. *Journal of Social Issues*, 66(2):387–400, 2010.
- [135] Zimitri Erasmus. *Race otherwise: forging a new humanism for South Africa*. NYU Press, 2017.
- [136] Martin Erwig and Bernd Meyer. Heterogeneous visual languages-integrating visual and textual programming. In *Proceedings of Symposium on Visual Languages*, pages 318–325. IEEE, 1995.
- [137] Arturo Escobar. *Designs for the pluriverse: Radical interdependence, autonomy, and the making of worlds*. Duke University Press, 2018.
- [138] Maya Escueta, Vincent Quan, Andre Joshua Nickow, and Philip Oreopoulos. Education technology: An evidence-based review. Working Paper 23744, National Bureau of Economic Research, August 2017.
- [139] Matthias Felleisen. On the expressive power of programming languages. *Science of computer programming*, 17(1-3):35–75, 1991.
- [140] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi, Eli Barzilay, Jay McCarthy, and Sam Tobin-Hochstadt. The Racket Manifesto. In *1st Summit on Advances in Programming Languages*

(SNAPL 2015), volume 32 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 113–128, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

- [141] Barbara J Fields. Whiteness, racism, and identity. *International Labor and Working-Class History*, 60:48–56, 2001.
- [142] Karen E. Fields and Barbara J. Fields. *Racecraft: The soul of inequality in American life*. Verso Trade, 2014.
- [143] Dale Fisk. *Programming with punched cards*. Columbia University, Computing History Archives, 2005.
- [144] Jay Forrester et al. Reports of mit project whirlwind (summary report no. 33). Technical report, MIT Digital Computer Laboratory, 1953.
- [145] CS4All San Francisco. MyCS Quarter-Length Course, January 2020.
- [146] Nancy Fraser. From redistribution to recognition? dilemmas of justice in a ‘post-socialist’ age. *New Left Review*, (212):68, 1995.
- [147] Gottlob Frege. *Begriffsschrift: eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Halle, 1879.
- [148] Paulo Freire. *Pedagogy of the oppressed*. 1970.
- [149] Craig Gidney. Quirk: Quantum circuit simulator, 2017. <https://algassert.com/quirk>.
- [150] Howard Giles and Bernadette Watson. Intercultural and intergroup communication. *International encyclopedia of communication*, 6:2337–2348, 2008.
- [151] W. K. Giloi. Konrad zuse’s plankalkül: the first high-level, “non-von neumann” programming language. *IEEE Annals of the History of Computing*, 19(2):17–24, April 1997.
- [152] Paul Gilroy. *Against race: Imagining political culture beyond the color line*. Harvard University Press, 2000.
- [153] Paul Gilroy. *Postcolonial melancholia*. Columbia University Press, 2005.

- [154] Paul Gilroy. True humanism? civilisationism, securitocracy and racial resignation. In *Johannesburg Workshop in Theory and Criticism Salon*, volume 1, 2009.
- [155] Paul Gilroy. Paul gilroy in search of a not necessarily safe starting point. *Open Democracy*, 2016.
- [156] Paul Gilroy. Never again: Refusing race and salvaging the human, 2019. University of Bergen. Transcript of 2019 Holberg Lecture <https://www.youtube.com/watch?v=Ta6UkmlXtVo>.
- [157] Paul Gilroy and David Theo Goldberg. In conversation with david theo goldberg, 2020. Interview transcript.
- [158] Paul Gilroy and Adam Schatz. The absurdities of race, 2020. Audio interview. Transcript provided at link.
- [159] Paul Gilroy and Patricia J. Williams. In conversation with patricia j. williams, 2020. Interview transcript.
- [160] Eddie S Glaude. *Begin again: James Baldwin's America and its urgent lessons for our own*. Crown Books, 2020.
- [161] David Theo Goldberg. *Are we all postracial yet?* John Wiley & Sons, 2015.
- [162] Adele Goldstine and John von Neumann. Monte carlo flow diagram, Dec. 4, 1947. John von Neumann Papers, Library of Congress Manuscripts Division box 12, folder 6.
- [163] Herman H. Goldstine. *The computer from Pascal to von Neumann*. Princeton University Press, 1993.
- [164] Herman H. Goldstine and John Von Neumann. Planning and coding of problems for an electronic computing instrument. Technical report, Moore School of Electrical Engineering, University of Pennsylvania, 1947.
- [165] Google. People + AI Guidebook, 2019. <https://pair.withgoogle.com/guidebook/>.
- [166] Google Quantum AI. Cirq, 2022. <https://quantumai.google/cirq>.

- [167] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper: A scalable quantum programming language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13*, pages 333–342, New York, NY, USA, 2013. Association for Computing Machinery.
- [168] Thomas RG Green and Marian Petre. When visual programs are harder to read than textual programs. In *Human-Computer Interaction: Tasks and Organisation, Proceedings ECCE-6 (6th European Conference Cognitive Ergonomics)*, volume 57. Citeseer, 1992.
- [169] Saul Greenberg and Bill Buxton. Usability evaluation considered harmful (some of the time). In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '08*, pages 111–120, New York, NY, USA, 2008. Association for Computing Machinery.
- [170] Anthony G Greenwald and Mahzarin R Banaji. Implicit social cognition: attitudes, self-esteem, and stereotypes. *Psychological review*, 102(1):4, 1995.
- [171] Shayla R Griffin, Mikel Brown, and Naomi M Warren. Critical education in high schools: The promise and challenges of intergroup dialogue. *Equity & Excellence in Education*, 45(1):159–180, 2012.
- [172] Mark D. Gross and Ellen Yi-Luen Do. Ambiguous intentions: A paper-like interface for creative design. In *Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology, UIST '96*, pages 183–192, New York, NY, USA, 1996. Association for Computing Machinery.
- [173] Sumit Gulwani, Oleksandr Polozov, Rishabh Singh, et al. Program synthesis. *Foundations and Trends® in Programming Languages*, 4(1-2):1–119, 2017.
- [174] Philip J. Guo. Non-native english speakers learning computer programming: Barriers, desires, and design opportunities. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI '18*, New York, NY, USA, 2018. ACM.
- [175] Patricia Gurin, Biren Ratnesh A. Nagda, and Ximena Zuniga. *Dialogue across difference: Practice, theory, and research on intergroup dialogue*. Russell Sage Foundation, 2013.
- [176] Kris D Gutiérrez, Patricia Baquedano-López, and Carlos Tejeda. Rethink-

ing diversity: Hybridity and hybrid language practices in the third space. *Mind, culture, and activity*, 6(4):286–303, 1999.

- [177] Kris D. Gutiérrez, P. Zitlali Morales, and Danny C. Martinez. Remediating literacy: Culture, difference, and learning for students from nondominant communities. *Review of research in education*, 33(1):212–245, 2009.
- [178] Mark Guzdial. We should stop saying ‘language independent.’ we don’t know how to do that. *Communications of the ACM Blog*, Aug 2019.
- [179] Asad Haider. *Mistaken identity: Race and class in the age of Trump*. Verso Books, 2018. eBook. Page number corresponds to PDF.
- [180] Thomas Haigh. Actually, turing did not invent the computer. *Communications of the ACM*, 57(1):36–41, 2014.
- [181] Thomas Haigh and Mark Priestley. Where code comes from: Architectures of automatic control from babbage to algol. *Communications of the ACM*, 59(1):39–44, 2015.
- [182] Thomas Haigh, Peter Mark Priestley, Mark Priestley, and Crispin Rope. *ENIAC in action: Making and remaking the modern computer*. MIT Press, 2016.
- [183] Edward T. Hall. *The Silent Language*. Anchor books, 1959.
- [184] Stuart Hall. Who needs identity. *Questions of cultural identity*, 16(2):1–17, 1996.
- [185] Amber M. Hamilton. What’s missing from corporate statements on racial injustice? the real cause of racism. *MIT Technology Review*, September 2020.
- [186] Mitchell R. Hammer and Milton Bennett. The intercultural development inventory. *Student learning abroad*, pages 115–136, 2012.
- [187] Tracy Hammond and Randall Davis. Ladder, a sketching language for user interface developers. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH ’07, pages 35–es, New York, NY, USA, 2007. Association for Computing Machinery.

- [188] David Hankerson, Andrea R Marshall, Jennifer Booker, Houda El Mismouni, Imani Walker, and Jennifer A Rode. Does technology have race? In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 473–486, 2016.
- [189] Alex Hanna, Emily Denton, Andrew Smart, and Jamila Smith-Loud. Towards a critical race methodology in algorithmic fairness. In *Proceedings of the 2020 conference on fairness, accountability, and transparency*, pages 501–512, 2020.
- [190] Rebecca L. Hao and Elena L. Glassman. Approaching polyglot programming: what can we learn from bilingualism studies? 2019.
- [191] Rebecca L. Hao and Elena L. Glassman. Approaching Polyglot Programming: What Can We Learn from Bilingualism Studies? In Sarah Chasins, Elena L. Glassman, and Joshua Sunshine, editors, *10th Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU 2019)*, volume 76 of *OpenAccess Series in Informatics (OASICs)*, pages 1:1–1:7, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [192] Donna Haraway. Situated knowledges: The science question in feminism and the privilege of partial perspective. *Feminist studies*, 14(3):575–599, 1988.
- [193] Steve Harrison, Phoebe Sengers, and Deborah Tatar. Making epistemological trouble: Third-paradigm hci as successor science. *Interacting with Computers*, 23(5):385–392, 2011.
- [194] Sally Haslanger. Gender and race:(what) are they?(what) do we want them to be? *Noûs*, 34(1):31–55, 2000.
- [195] Andrew Head, Fred Hohman, Titus Barik, Steven M. Drucker, and Robert DeLine. Managing messes in computational notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI '19*, pages 1–12, New York, NY, USA, 2019. Association for Computing Machinery.
- [196] Brian Hempel, Justin Lubin, and Ravi Chugh. Sketch-n-sketch: Output-directed programming for svg. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology, UIST '19*, pages 281–292, New York, NY, USA, 2019. Association for Computing Machinery.

- [197] Barnor Hesse. Self-fulfilling prophecy: The postracial horizon. *South Atlantic Quarterly*, 110(1):155–178, 2011.
- [198] David Hilbert and Wilhelm Ackerman. *Grundzüge der Theoretischen Logik*. Springer-Verlag, 2 edition, 1928.
- [199] Lawrence A. Hirschfeld. *Race in the making: Cognition, culture, and the child's construction of human kinds*. MIT Press, 1998.
- [200] Andrew Hodges. *Alan Turing: The Enigma*. Random House, 2012.
- [201] Geert Hofstede. Dimensionalizing cultures: The hofstede model in context. *Online readings in psychology and culture*, 2(1):2307–0919, 2011.
- [202] Frances E. Holberton. Personal interview. 14 apr. 1983, oh 50. oral history interview by james baker ross, potomac, maryland. charles babbage institute. *University of Minnesota, Minneapolis*, 1983.
- [203] Paula K. Hooper. They have their own thoughts. *Constructionism in practice: Designing, thinking, and learning in a digital world*, page 241, 1996.
- [204] Rebecca Horn. Responses to intimate partner violence in kakuma refugee camp: refugee interactions with agency systems. *Social science & medicine*, 70(1):160–168, 2010.
- [205] Carlos A. Hoyt. The pedagogy of the meaning of racism: Reconciling a discordant discourse. *Social work*, 57(3):225–234, 2012.
- [206] Carlos A. Hoyt. *The Arc of a Bad Idea: Understanding and Transcending Race*. Oxford University Press, 2016.
- [207] Carlos A. Hoyt. How not to talk about race: Breaking the racialization habit to overcome racism, June 2019. Presentation given at MCLA DEI Conference. Relevant quote: “Racial social identity adjectives (e.g. ‘black’ and ‘white’) will be replaced by ‘black-racialized’ and ‘white-racialized.’”.
- [208] Ellen Huet. Camp grounded: Where people pay \$570 to have their smartphones taken away from them. *Forbes*, Jun 2014.
- [209] Edwin L. Hutchins, James D. Hollan, and Donald A. Norman. Direct manipulation interfaces. *Human-computer interaction*, 1(4):311–338, 1985.

- [210] IBM. Qiskit textbook: Learn quantum computation using Qiskit, 2022. <https://qiskit.org/textbook/preface.html>.
- [211] Noel Ignatiev. The point is not to interpret whiteness but to abolish it. *Race Traitor*, 1997.
- [212] Noel Ignatiev and John Garvey. Abolish the white race by any means necessary. *Race Traitor*, 1, 1993.
- [213] Tim Ingold. *Making: Anthropology, archaeology, art and architecture*. Routledge, 2013.
- [214] Tim Ingold. *Lines: a brief history*. Routledge, 2016.
- [215] Lilly C. Irani and Paul Dourish. Postcolonial interculturality. In *Proceedings of the 2009 International Workshop on Intercultural Collaboration, IWIC '09*, page 249–252, New York, NY, USA, 2009. ACM.
- [216] Kenneth Iverson. Notation as a tool of thought, acm turing award lecture, 1979. acm turing award lectures, the first twenty years, 1987.
- [217] Steven J. Jackson and Sarah Barbrow. Standards and/as innovation: Protocols, creativity, and interactive systems development in ecology. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI '15*, pages 1769–1778, New York, NY, USA, 2015. ACM.
- [218] Bastian Jaeger and Willem Sleegers. Racial discrimination in the sharing economy: Evidence from airbnb markets across the world. 2020.
- [219] Sheila Jasanoff et al. *States of knowledge: the co-production of science and the social order*. Routledge, 2004.
- [220] Bernward Joerges. Do politics have artefacts? *Social studies of science*, 29(3):411–431, 1999.
- [221] Gabe Johnson, Mark D Gross, Jason Hong, and Ellen Yi-Luen Do. Computational support for sketching in design: A review. *Human-Computer Interaction*, 2(1):1–93, 2008.
- [222] John L. Jones. *A survey of automatic coding techniques for digital computers*. PhD thesis, Massachusetts Institute of Technology, 1954.

- [223] Yasmin Kafai. Constructionist visions: Hard fun with serious games. *International Journal of Child-Computer Interaction*, 2018.
- [224] Yasmin Kafai, Kristin Searle, Cr stobal Martinez, and Bryan Brayboy. Ethnocomputing with electronic textiles: Culturally responsive open design to broaden participation in computing in american indian youth and communities. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 241–246, 2014.
- [225] David Kaiser. *Drawing theories apart*. University of Chicago Press, 2009.
- [226] DaYe Kang, Tony Ho, Nicolai Marquardt, Bilge Mutlu, and Andrea Bianchi. Toonnote: Improving communication in computational notebooks using interactive data comics. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, CHI ’21*, New York, NY, USA, 2021. Association for Computing Machinery.
- [227] Laewoo Kang and Steven Jackson. Tech-art-theory: Improvisational methods for hci learning and teaching. *Proc. ACM Hum.-Comput. Interact.*, 5(CSCW1), Apr 2021.
- [228] Victor Kaptelinin and Bonnie A. Nardi. *Acting with technology: Activity theory and interaction design*. MIT Press, 2006.
- [229] Naveena Karusala, Aditya Vishwanath, Arkadeep Kumar, Aman Mangal, and Neha Kumar. Care as a resource in underserved learning environments. *Proceedings of the ACM on Computer-Supported Collaborative Work*, 1:1–22, 2017.
- [230] Atoosa Kasirzadeh and Andrew Smart. The use and misuse of counterfactuals in ethical machine learning. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 228–236, 2021.
- [231] Yuko Kawai. Deracialised race, obscured racism: Japaneseness, western and japanese concepts of race, and modalities of racism. *Japanese Studies*, 35(1):23–47, 2015.
- [232] Alan Kay. Doing with images makes symbols: Communicating with computers, 1987. University Video Communications. Film.
- [233] Ibram X Kendi. *How to be an Antiracist*. One World/Ballantine, 2019.

- [234] UNHCR Kenya. Kakuma Refugee Camp and Kalobeyei Integrated Settlement, 2018.
- [235] Mary Beth Kery, Donghao Ren, Fred Hohman, Dominik Moritz, Kanit Wongsuphasawat, and Kayur Patel. Mage: Fluid moves between code and graphical work in computational notebooks. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, pages 140–151, New York, NY, USA, 2020. Association for Computing Machinery.
- [236] Zaid Khan and Yun Fu. One label, one billion faces: Usage and consistency of racial categories in computer vision. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 587–597, 2021.
- [237] Vera Khovanskaya, Lynn Dombrowski, Ellie Harmon, Matthias Korn, Ann Light, Michael Stewart, and Amy Volda. Designing against the status quo. *interactions*, 25(2), 2018.
- [238] Young Yun Kim. The identity factor in intercultural competence. *The Sage handbook of intercultural competence*, 1:53–65, 2009.
- [239] Friedrich A. Kittler. *Gramophone, film, typewriter*. Stanford University Press, 1999.
- [240] Jon Kleinberg, Jens Ludwig, Sendhil Mullainathan, and Ashesh Rambachan. Algorithmic fairness. In *Aea papers and proceedings*, volume 108, pages 22–27, 2018.
- [241] Donald E. Knuth and Luis Trabb Pardo. The early development of programming languages. In *A history of computing in the twentieth century*, pages 197–273. Elsevier, 1980.
- [242] Allison Koenecke, Andrew Nam, Emily Lake, Joe Nudell, Minnie Quartey, Zion Mengesha, Connor Touns, John R Rickford, Dan Jurafsky, and Sharad Goel. Racial disparities in automated speech recognition. *Proceedings of the National Academy of Sciences*, 117(14):7684–7689, 2020.
- [243] Philip Kraft. *Programmers and managers: The routinization of computer programming in the United States*. Springer Science & Business Media, 2012.
- [244] paperson la. A third university is possible, 2017. (Sorry, had to cite you).

- [245] James A. Landay and Brad A. Myers. Interactive sketching for the early stages of user interface design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '95*, pages 43–50, USA, 1995. ACM Press/Addison-Wesley Publishing Co.
- [246] James A. Landay and Brad A. Myers. Sketching interfaces: toward more human interface design. *Computer*, 34(3):56–64, 2001.
- [247] David Landy and Robert L. Goldstone. Formal notations are diagrams: Evidence from a production task. *Memory & cognition*, 35(8):2033–2040, 2007.
- [248] Bruno Latour. Reassembling the social: an introduction to actor-network-theory. *Oxford University Press*, 2005.
- [249] Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. Deduplicating training data makes language models better. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8424–8445, 2022.
- [250] Wendy Leeds-Hurwitz. Notes in the history of intercultural communication: The foreign service institute and the mandate for intercultural training. 1990.
- [251] Zeus Leonardo. *Race frameworks: A multidimensional theory of racism and education*. Teachers College Press, 2013.
- [252] Jaime Lester, Aoi Yamanaka, and Brice Struthers. Gender microaggressions and learning environments: The role of physical space in teaching pedagogy and communication. *Community College Journal of Research and Practice*, 40(11):909–926, 2016.
- [253] Meira Levinson. *No citizen left behind*, volume 13. Harvard University Press, 2012.
- [254] Colleen M Lewis. How programming environment shapes perception, learning and goals: logo vs. scratch. In *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 346–350, 2010.
- [255] Colleen M. Lewis and Niral Shah. How equity and inequity can emerge

- in pair programming. In *Proceedings of the eleventh annual international conference on international computing education research*, pages 41–50, 2015.
- [256] Chuanjun Li, Timothy S Miller, Robert C Zeleznik, and Joseph J LaViola Jr. Algorsketch: Algorithm sketching and interactive computation. In *SBM*, pages 175–182. Citeseer, 2008.
- [257] Henry Lieberman. The tyranny of evaluation. *ACM CHI Fringe*, 2003.
- [258] Jennifer S. Light. When computers were women. *Technology and culture*, 40(3):455–483, 1999.
- [259] Seongtaek Lim, Rama Adithya Varanasi, and Tapan Parikh. Glide (git-learning ide; integrated development environment): In-class collaboration in web engineering curriculum for youths (abstract only). In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education, SIGCSE '18*, pages 1112–1112, New York, NY, USA, 2018. ACM.
- [260] James Lin, Mark W. Newman, Jason I. Hong, and James A. Landay. Denim: Finding a tighter fit between tools and practice for web site design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '00*, pages 510–517, New York, NY, USA, 2000. Association for Computing Machinery.
- [261] Silvia Lindtner, Shaowen Bardzell, and Jeffrey Bardzell. Reconstituting the utopian vision of making: Hci after technosolutionism. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, CHI '16*, page 1390–1402, New York, NY, USA, 2016. ACM.
- [262] Silvia Lindtner, Shaowen Bardzell, and Jeffrey Bardzell. Design and intervention in the age of “no alternative”. *Proc. ACM Hum.-Comput. Interact.*, 2(CSCW), November 2018.
- [263] Gil Loescher and James Milner. The long road home: Protracted refugee situations in africa. *Survival*, 47(2):153–174, 2005.
- [264] Mara Loveman. Is “race” essential? *American Sociological Review*, 64(6):891–898, 1999.
- [265] Alex Lubin and Ussama Makdisi. Coexistence, sectarianism and racism – an interview with ussama makdisi. *Middle East Research and Information Project*, June 2021.

- [266] Peter Lyman. Reading, writing and word processing: Toward a phenomenology of the computer age. *Qualitative Sociology*, 7(1-2):75–89, 1984.
- [267] Zine Magubane. American sociology’s racial ontology: Remembering slavery, deconstructing modernity, and charting the future of global historical sociology. *Cultural Sociology*, 10(3):369–384, 2016.
- [268] Jabari Mahiri. *Deconstructing Race: Multicultural Education Beyond the Color-Blind*. Teachers College Press, 2017.
- [269] Sandra Manninen, Lauri Tuominen, Robin I Dunbar, Tomi Karjalainen, Jussi Hirvonen, Eveliina Arponen, Riitta Hari, Iiro P Jääskeläinen, Mikko Sams, and Lauri Nummenmaa. Social laughter triggers endogenous opioid release in humans. *Journal of Neuroscience*, 37(25):6125–6131, 2017.
- [270] Chuei D. Mareng. Reflections on refugee students’ major perceptions of education in Kakuma Refugee Camp, Kenya. *Intercultural Education*, 21(5):473–481, 2010.
- [271] Jane Margolis, Rachel Estrella, Joanna Goode, Kim Nao, and Jennifer Jellison Holme. *Stuck in the Shallow End: Education, Race, and Computing*. MIT Press, 2008.
- [272] Joel Eduardo Martinez and Elizabeth Levy Paluck. Quantifying shared and idiosyncratic judgments of racism in social discourse. 2020.
- [273] Sheena Michele Mason. *Decolonizing the Raci(al/st) Imagination in Literary Studies*. PhD thesis, Howard University, 2021.
- [274] David Matsumoto and Hyisung C Hwang. Assessing cross-cultural competence: A review of available tests. *Journal of cross-cultural psychology*, 44(6):849–873, 2013.
- [275] Jacob Matthews and Robert Bruce Findler. Operational semantics for multi-language programs. *ACM Trans. Program. Lang. Syst.*, 31(3), April 2009.
- [276] Charlton D. McIlwain. *Black software: The internet and racial justice, from the AfroNet to Black Lives Matter*. Oxford University Press, USA, 2019.
- [277] Brad A. Meisner. Are you OK, Boomer? Intensification of ageism and in-

- tergenerational tensions on social media amid COVID-19. *Leisure Sciences*, pages 1–6, 2020.
- [278] Edward F. Melcer and Katherine Isbister. Bots & (main)frames: Exploring the impact of tangible blocks and collaborative play in an educational programming game. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, New York, NY, USA, 2018. ACM.
- [279] Richard L. Mendelsohn. *The Philosophy of Gottlob Frege*. Cambridge University Press, 2005.
- [280] Mary Mendenhall, Sarah Dryden-Peterson, Lesley Bartlett, Caroline Ndirangu, Rosemary Imonje, Daniel Gakunga, Loise Gichuhi, Grace Nyagah, Ursulla Okoth, and Mary Tangelder. Quality education for refugees in kenya: Pedagogy in urban nairobi and kakuma refugee camp settings. 2015.
- [281] Robert K. Merton. The self-fulfilling prophecy. *The antioch review*, 8(2):193–210, 1948.
- [282] Microsoft. Sketch2code, 2018. <https://www.microsoft.com/en-us/ai/ai-lab-sketch2code>.
- [283] Marie Moran. *Identity and capitalism*. Sage, 2014.
- [284] Cueponcaxochitl Dianna Moreno Sandoval. Critical ancestral computing: A culturally relevant computer science education. *PsychNology Journal*, 11(1), 2013.
- [285] Stephen J. Morris and OCZ Gotel. Flow diagrams: rise and fall of the first software engineering notation. In *International Conference on Theory and Application of Diagrams*, pages 130–144. Springer, 2006.
- [286] Toni Morrison. Home. In Wahneema Lubiano, editor, *The House That Race Built*, chapter 1. Pantheon, 1997.
- [287] Toni Morrison. Home. In *The House that Race Built: Original essays by Toni Morrison, Angela Y. Davis, Cornel West, and others on Black Americans and politics in America today*. Vintage, 1997.
- [288] Toni Morrison. *The origin of others*. Harvard University Press, 2017.

- [289] Dhawal Mujumdar, Manuel Kallenbach, Brandon Liu, and Björn Hartmann. Crowdsourcing suggestions to programming problems for dynamic web development languages. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, pages 1525–1530, New York, NY, USA, 2011. Association for Computing Machinery.
- [290] Thomas S. Mullaney. *The Chinese typewriter: A history*. MIT Press, 2017.
- [291] Kyle Muzyka. A Hawaiian team’s mission to translate programming language to their Native language. 2018.
- [292] Jennifer C Nash. Re-thinking intersectionality. *Feminist review*, 89(1):1–15, 2008.
- [293] Jennifer C. Nash. *Black feminism reimaged: After intersectionality*. Duke University Press, 2018.
- [294] Na’ilah Suad Nasir and Sepehr Vakil. Stem-focused academies in urban schools: Tensions and possibilities. *Journal of the Learning Sciences*, 26(3):376–406, 2017.
- [295] Ramsey Nasser. *Qalb*, 2012.
- [296] United Nations. *Sustainable Development Goals Report 2016*. UN, 2016.
- [297] Joe Navarro and Marvin Karlins. *What every body is saying*. HarperCollins, 2016.
- [298] Anoop Nayak. After race: Ethnography, race and post-race theory. *Ethnic and racial studies*, 29(3):411–430, 2006.
- [299] Hans Neukom. Ermeth: The first swiss computer. *IEEE Annals of the History of Computing*, 27(4):5–22, 2005.
- [300] Sasha Newell. Decolonizing science, digitizing the occult: Theory from the virtual south. *African Studies Review*, 64(1):86–104, 2021.
- [301] Safiya Umoja Noble. *Algorithms of oppression: How search engines reinforce racism*. NYU Press, 2018.

- [302] David Nofre, Mark Priestley, and Gerard Alberts. When technology became language: The origins of the linguistic conception of computer programming, 1950–1960. *Technology and Culture*, 55(1):40–75, 2014.
- [303] Osagie Obasogie. *Blinded by sight: Seeing race through the eyes of the blind*. Stanford University Press, 2013.
- [304] Theresa O’Connell, Chuanjun Li, Timothy S. Miller, Robert C. Zeleznik, and Joseph J. LaViola. A usability evaluation of algosketch: A pen-based application for mathematics. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling, SBIM ’09*, pages 149–157, New York, NY, USA, 2009. Association for Computing Machinery.
- [305] Ihudiya Finda Ogbonnaya-Ogburu, Angela DR Smith, Alexandra To, and Kentaro Toyama. Critical race theory for HCI. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–16, 2020.
- [306] Cyrus Omar and Jonathan Aldrich. Reasonably programmable literal notation. *Proc. ACM Program. Lang.*, 2(ICFP), Jul 2018.
- [307] Cyrus Omar, David Moon, Andrew Blinn, Ian Voysey, Nick Collins, and Ravi Chugh. Filling typed holes with live guis. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2021*, pages 511–525, New York, NY, USA, 2021. Association for Computing Machinery.
- [308] Cyrus Omar, Ian Voysey, Ravi Chugh, and Matthew A. Hammer. Live functional programming with typed holes. *Proc. ACM Program. Lang.*, 3(POPL), Jan 2019.
- [309] Michael Omi and Howard Winant. *Racial formation in the United States*. Routledge, 2014.
- [310] Walter J. Ong. *Orality and literacy*. Routledge, 2013.
- [311] Wanda J. Orlikowski. Integrated information environment or matrix of control? the contradictory implications of information technology. *Accounting, Management and Information Technologies*, 1(1):9–42, 1991.
- [312] Nell Irvin Painter. *The history of white people*. WW Norton & Company, 2010.

- [313] Elizabeth Levy Paluck and Donald P Green. Prejudice reduction: What works? a review and assessment of research and practice. *Annual review of psychology*, 60:339–367, 2009.
- [314] Seymour Papert. *Mindstorms: children, computers, and powerful ideas*. Basic Books, 1980.
- [315] V.D. Parondzhanov. Visual syntax of the drakon language. *Programming and Computer Software*, 21(3), 1995.
- [316] Harold Pashler, Mark McDaniel, Doug Rohrer, and Robert Bjork. Learning styles: Concepts and evidence. *Psychological science in the public interest*, 9(3):105–119, 2008.
- [317] Samir Passi and Steven Jackson. Data vision: Learning to see through algorithmic abstraction. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing, CSCW '17*, page 2436–2447, New York, NY, USA, 2017. ACM.
- [318] Samir Passi and Steven J. Jackson. Trust in data science: Collaboration, translation, and accountability in corporate data science projects. *Proc. ACM Hum.-Comput. Interact.*, 2(CSCW), November 2018.
- [319] Leigh Patel and Alton Price. The origins, potentials, and limits of racial justice. *Critical Ethnic Studies*, 2(2):61–81, 2016.
- [320] Orlando Patterson. *The ordeal of integration: Progress and resentment in America's "racial" crisis*. Civitas/Counterpoint Washington, DC, 1997.
- [321] Orlando Patterson. Making sense of culture. *Annual Review of Sociology*, 40:1–30, 2014.
- [322] Joshua Paul. Post-racial futures: imagining post-racialist anti-racism (s). *Ethnic and Racial Studies*, 37(4):702–718, 2014.
- [323] Sara Pavanello, Samir Elhawary, and Sara Pantuliano. Hidden and exposed: Urban refugees in nairobi, kenya. Technical report, Overseas Development Institute London, 2010.
- [324] Udai Singh Pawar, Joyojeet Pal, and Kentaro Toyama. Multiple mice for computers in education in developing countries. In *Proceedings of the Infor-*

mation and Communication Technologies and Development (ICTD) Conference, pages 64–71. IEEE, 2006.

- [325] Jennifer Paykin, Robert Rand, and Steve Zdancewic. Qwire: A core language for quantum circuits. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL '17*, pages 846–858, New York, NY, USA, 2017. Association for Computing Machinery.
- [326] Norman Peitek, Janet Siegmund, Sven Apel, Christian Kästner, Chris Parnin, Anja Bethmann, Thomas Leich, Gunter Saake, and André Brechmann. A look into programmers' heads. *IEEE Transactions on Software Engineering*, 2018.
- [327] Roger Penrose. Applications of negative dimensional tensors. *Combinatorial mathematics and its applications*, 1:221–244, 1971.
- [328] Thomas F. Pettigrew. Intergroup contact theory. *Annual review of psychology*, 49(1):65–85, 1998.
- [329] Cornel Pewewardy. Renaming ourselves on our own terms: Race, tribal nations, and representation in education. 2000.
- [330] Kavita Philip, Lilly Irani, and Paul Dourish. Postcolonial computing: A tactical survey. *Science, Technology, & Human Values*, 37(1):3–29, 2012.
- [331] Andrew Pickering. *Science as Practice and Culture*. University of Chicago Press, 1992.
- [332] Andrew Pickering. *The Mangle of Practice: Time, Agency, and Science*. University of Chicago Press, 1995.
- [333] Nichole Pinkard, Sheena Erete, Caitlin K. Martin, and Maxine McKinney de Royston. Digital youth divas: Exploring narrative-driven curriculum to spark middle school girls' interest in computational activities. *Journal of the Learning Sciences*, 26(3):477–516, 2017.
- [334] Adrian Piper. Rationality and the structure of the self volume ii: A kantian conception. 2013.
- [335] Adrian Piper. *Escape to Berlin: A Travel Memoir*. Adrian Piper Research Archive Foundation Berlin, 2018.

- [336] Adrian Piper. Df. racial essentialism, May 2019.
- [337] Adrian Piper and Agata Waleczek. ‘i still do believe they want me dead’: An interview with adrian piper. *Frieze Magazine*, Sept 2018.
- [338] John Anthony Powell. *Racing to justice: Transforming our conceptions of self and other to build an inclusive society*. Indiana University Press, 2012.
- [339] Michael Powell. A Black Marxist Scholar Wanted to Talk About Race. It Ignited a Fury. *The New York Times*, August 2020.
- [340] Mark Priestley. The mathematical origins of modern computing. In *Technology and Mathematics*, pages 107–135. Springer, 2018.
- [341] Mark Priestley. *Routines of Substitution: John von Neumann’s Work on Software Development, 1945–1948*. Springer, 2018.
- [342] Jasbir K. Puar. “i would rather be a cyborg than a goddess”: Becoming-intersectional in assemblage theory. *PhiloSOPHIA*, 2(1):49–66, 2012.
- [343] Jasbir K. Puar. *Terrorist assemblages: Homonationalism in queer times*. Duke University Press, 2018.
- [344] Yolanda A. Rankin and Jakita O. Thomas. Straighten up and fly right: rethinking intersectionality in hci research. *interactions*, 26(6):64–68, 2019.
- [345] Yolanda A. Rankin, Jakita O. Thomas, and Sheena Erete. Real talk: Saturated sites of violence in cs education. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pages 802–808, 2021.
- [346] John Rawls. *A Theory of Justice: Revised Edition*. Harvard University Press, 1999.
- [347] Jenny Reardon. *Race to the Finish*. Princeton University Press, 2009.
- [348] Toure Reed. *Toward Freedom: The Case Against Race Reductionism*. Verso Books, 2020.
- [349] Adolph Reed Jr. Marx, race, and neoliberalism. In *New Labor Forum*, volume 22, pages 49–57. SAGE Publications Sage CA: Los Angeles, CA, 2013.

- [350] Adolph Reed Jr. Antiracism: a neoliberal alternative to a left. *Dialectical Anthropology*, 42(2):105–115, 2018.
- [351] Adolph Reed Jr and Merlin Chowkwanyun. Race, class, crisis: The discourse of racial disparity and its analytical discontents. *Socialist Register*, 48:149–175, 2012.
- [352] Remington Rand. Remington-Rand Presents the UNIVAC, 1952. Computer History Museum. Film.
- [353] Mathys Rennela and Sam Staton. Classical control, quantum circuits and linear logic in enriched category theory. *CoRR*, abs/1711.05159, 2017.
- [354] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.
- [355] Lauren Rhue. Racial influence on automated perceptions of emotions. Available at SSRN 3281765, 2018.
- [356] Raúl Rojas, Cüneyt Göktekin, Gerald Friedland, Mike Krüger, Olaf Langmack, and Denis Kuniß. Plankalkül: The first high-level programming language and its implementation. *Freie Universität Berlin*, 2000.
- [357] Jeremy Roschelle and Stephanie D Teasley. The construction of shared knowledge in collaborative problem solving. In *Computer supported collaborative learning*, pages 69–97. Springer, 1995.
- [358] Daniela K. Rosner. *Critical Fabulations: Reworking the Methods and Margins of Design*. MIT Press, 2018.
- [359] Daniela K. Rosner, Samantha Shorey, Brock R. Craft, and Helen Remick. Making core memory: Design inquiry into gendered legacies of engineering and craftwork. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, New York, NY, USA, 2018. ACM.
- [360] Adam Rule, Aurélien Tabard, and James D. Hollan. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 1–12, New York, NY, USA, 2018. Association for Computing Machinery.

- [361] John G. Russell. Consuming passions: Spectacle, self-transformation, and the commodification of blackness in japan. *positions: east asia cultures critique*, 6(1):113–177, 1998.
- [362] John G. Russell. Replicating the white self and other: Skin color, racelessness, gynoids, and the construction of whiteness in japan. *Japanese studies*, 37(1):23–48, 2017.
- [363] Steve Russell. Apples are the color of blood. *Critical Sociology*, 28(1-2):65–76, 2002.
- [364] Heinz Rutishauser. Automatische rechenplanfertigung bei programmgesteuerten rechenmaschinen. *Zeitschrift für angewandte Mathematik und Physik ZAMP*, 3(4):312–313, 1952.
- [365] Jean J. Ryoo. Pedagogy that supports computer science for all. *ACM Transactions on Computing Education (TOCE)*, 19(4):1–23, 2019.
- [366] Jean J. Ryoo and Linda Kekelis. Reframing “failure” in making: The value of play, social relationships, and ownership. *Journal of Youth Development*, 13(4):49–67, 2018.
- [367] Jean J. Ryoo, Tiera Tanksley, Cynthia Estrada, and Jane Margolis. Take space, make space: how students use computer science to disrupt and resist marginalization in schools. *Computer Science Education*, 30(3):337–361, 2020.
- [368] Mark Santolucito. Human-in-the-loop program synthesis for live coding. In *Proceedings of the 9th ACM SIGPLAN International Workshop on Functional Art, Music, Modelling, and Design*, FARM 2021, pages 47–53, New York, NY, USA, 2021. Association for Computing Machinery.
- [369] Larry Saphire. Interview with john w. backus, Dec. 15, 1967. John W. Backus Papers, Library of Congress Manuscripts Division, box 1, item 7.
- [370] Nazmus Saquib et al. *Embodied mathematics by interactive sketching*. PhD thesis, Massachusetts Institute of Technology, 2020.
- [371] Nazmus Saquib, Rubaiat Habib Kazi, Li-yi Wei, Gloria Mark, and Deb Roy. Constructing embodied algebra by sketching. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI ’21, New York, NY, USA, 2021. Association for Computing Machinery.

- [372] Toby Schachman. “we’ve been playing with editing printed code by just pointing a keyboard at a page and typing.” tweet, March 1, 2018.
- [373] Ari Schlesinger, W Keith Edwards, and Rebecca E Grinter. Intersectional hci: Engaging identity through gender, race, and class. In *Proceedings of the 2017 CHI conference on human factors in computing systems*, pages 5412–5427, 2017.
- [374] Ari Schlesinger, Kenton P O’Hara, and Alex S Taylor. Let’s talk about race: Identity, chatbots, and ai. In *Proceedings of the 2018 chi conference on human factors in computing systems*, pages 1–14, 2018.
- [375] Tony Scott, Michael Cole, and Martin Engel. Chapter 5: Computers and education: A cultural constructivist perspective. *Review of research in education*, 18(1):191–251, 1992.
- [376] William H. Sewell et al. The concept (s) of culture. In *Practicing history*, pages 90–110. Routledge, 2004.
- [377] Rainier Spencer. *Reproducing race: The paradox of generation mix*. Lynne Rienner Publishers Boulder, CO, 2011.
- [378] Rainier Spencer. *Spurious issues: Race and multiracial identity politics in the United States*. Routledge, 2019.
- [379] Rainier Spencer and Alex Barnett. The sociology and real-ness of race, 2020. On the Multi-racial Family Man podcast. Audio interview.
- [380] Gayatri Chakravorty Spivak. Subaltern studies: Deconstructing historiography. In *In other worlds*, pages 270–304. Routledge, 2012.
- [381] Susan Leigh Star and Karen Ruhleder. Steps toward an ecology of infrastructure: Design and access for large information spaces. *Information systems research*, 7(1):111–134, 1996.
- [382] Walter G. Stephan and Cookie White Stephan. Predicting prejudice. *International Journal of Intercultural Relations*, 20(3-4):409–426, 1996.
- [383] Michèle Stephenson and Brian Young. A conversation with native americans on race. New York Times. Video.

- [384] Valerie Strauss. Students protest zuckerberg-backed digital learning program and ask him: ‘what gives you this right?’. *The Washington Post*, Nov 2018.
- [385] Sharon Hartman Strom. “light manufacturing”: The feminization of american office work, 1900–1930. *ILR Review*, 43(1):53–71, 1989.
- [386] Lucy Suchman. Located accountabilities in technology production. *Scandinavian journal of information systems*, 14(2):7, 2002.
- [387] Sarah Suleri, Vinoth Pandian Sermuga Pandian, Svetlana Shishkovets, and Matthias Jarke. Eve: A sketch-based software prototyping workbench. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI EA ’19, pages 1–6, New York, NY, USA, 2019. Association for Computing Machinery.
- [388] Sharifa Sultana, François Guimbretière, Phoebe Sengers, and Nicola Dell. Design within a patriarchal society: Opportunities and challenges in designing for rural women in bangladesh. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2018.
- [389] Joshua Sunshine, Elena Glassman, and Sarah Chasins. Plateau workshop, 2019.
- [390] Ivan E. Sutherland. *Sketchpad, a Man-Machine Graphical Communication System*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1963.
- [391] Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. Q# enabling scalable quantum computing and development with a high-level dsl. In *Proceedings of the Real World Domain Specific Languages Workshop 2018*, pages 1–10, 2018.
- [392] Henri Tajfel. Experiments in intergroup discrimination. *Scientific american*, 223(5):96–103, 1970.
- [393] Henri Tajfel. Social identity and intergroup behaviour. *Social science information*, 13(2):65–93, 1974.
- [394] Beverly Daniel Tatum. *Why are all the Black kids sitting together in the cafeteria?: And other conversations about race*. Basic Books, 2017.

- [395] George H. Taylor. Racism as the nation's crucial sin: Theology and derrick bell. *Mich. J. Race & L.*, 9:269, 2003.
- [396] Matti Tedre. *The science of computing: shaping a discipline*. Chapman and Hall/CRC, 2014.
- [397] The Black Experience Japan. "I'm Culturally Japanese..." (Black in Japan), 2019. Video interview.
- [398] The Black Experience Japan. "Our Kids Were Born And Raised in Japan..." (Black in Japan), 2019. Video interview.
- [399] Starlette Thomas. When we talk about race: Common misconceptions, Feb 2021.
- [400] Daniella Tilbury and Ingrid Mulà. Review of education for sustainable development policies from a cultural diversity and intercultural dialogue: Gaps and opportunities for future action. Technical report, UNESCO, 2009.
- [401] Alexandra To, Wenxia Sweeney, Jessica Hammer, and Geoff Kaufman. "they just don't get it": Towards social technologies for coping with interpersonal racism. *Proceedings of the ACM on Human-Computer Interaction*, 4(CSCW1):1–29, 2020.
- [402] Town of Princeville Residents. Town of princeville nc - our history, 2021.
- [403] Kentaro Toyama. *Geek heresy: Rescuing social change from the cult of technology*. PublicAffairs, 2015.
- [404] Jasper Tran O'Leary and Nadya Peek. Machine-o-matic: A programming environment for prototyping digital fabrication workflows. In *The Adjunct Publication of the 32nd Annual ACM Symposium on User Interface Software and Technology*, pages 134–136, 2019.
- [405] Eve Tuck and Julie Gorlewski. Racist ordering, settler colonialism, and edtpa: A participatory policy analysis. *Educational Policy*, 30(1):197–217, 2016.
- [406] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.

- [407] Sherry Turkle and Seymour Papert. Epistemological pluralism: Styles and voices within the computer culture. *Signs: Journal of women in culture and society*, 16(1):128–157, 1990.
- [408] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems, CHI EA '22*, New York, NY, USA, 2022. Association for Computing Machinery.
- [409] Sepehr Vakil. Ethics, identity, and political vision: Toward a justice-centered approach to equity in computer science education. *Harvard Educational Review*, 88(1):26–52, 2018.
- [410] Annette Vee. *Coding literacy: How computer programming is changing writing*. MIT Press, 2017.
- [411] Richard Velkley. The tension in the beautiful: On culture and civilization in rousseau and german philosophy. *Being after Rousseau: Philosophy and culture in question*, pages 11–30, 2002.
- [412] Helen Verran. *Science and an African logic*. University of Chicago Press, 2001.
- [413] Bret Victor, Paula Te, Josh Horowitz, Luke Iannini, Toby Schachman, and Virginia McArthur. *Dynamicland*, 2017.
- [414] John Von Neumann. *Papers of John von Neumann on computing and computer theory*, volume 12. MIT Press, 1987.
- [415] John Von Neumann. First draft of a report on the edvac. *IEEE Annals of the History of Computing*, 15(4):27–75, 1993.
- [416] Shirin Vossoughi, Paula K. Hooper, and Meg Escudé. Making through the lens of culture and power: Toward transformative visions for educational equity. *Harvard Educational Review*, 86(2):206–232, 2016.
- [417] Ngũgĩ Wa Thiong’o. *Moving the centre: The struggle for cultural freedoms*. James Currey, 1993.
- [418] Peter Wade. *Race and ethnicity in Latin America*. Pluto press, 2010.

- [419] April Yi Wang, Anant Mittal, Christopher Brooks, and Steve Oney. How data scientists use computational notebooks for real-time collaboration. *Proc. ACM Hum.-Comput. Interact.*, 3(CSCW), Nov 2019.
- [420] April Yi Wang, Dakuo Wang, Jaimie Drozdal, Michael Muller, Soya Park, Justin D. Weisz, Xuye Liu, Lingfei Wu, and Casey Dugan. Documentation matters: Human-centered ai system to assist data science code documentation in computational notebooks. *ACM Trans. Comput.-Hum. Interact.*, 29(2), Jan 2022.
- [421] Danli Wang, Yang Zhang, Tianyuan Gu, Liang He, and Hongan Wang. E-block: A tangible programming tool for children. In *Adjunct Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST Adjunct Proceedings '12, pages 71–72, New York, NY, USA, 2012. Association for Computing Machinery.
- [422] Vron Ware and Les Back. *Out of whiteness: Color, politics, and culture*. University of Chicago Press, 2002.
- [423] David Weintrop. Block-based programming in computer science education. *Commun. ACM*, 62(8):22–25, Jul 2019.
- [424] Linda L. Werner, Jill Denner, and Steven Bean. Pair programming strategies for middle school girls. In *CATE*, pages 161–166, 2004.
- [425] Kirsten N. Whitley. Visual programming languages and the empirical evidence for and against. *Journal of Visual Languages & Computing*, 8(1):109–142, 1997.
- [426] Isabel Wilkerson. *Caste: The Origins of Our Discontents*. Random House, 2020.
- [427] Thomas Chatterton Williams. *Self-Portrait in Black and White: Unlearning Race*. Hachette UK, 2019.
- [428] Tiffani L. Williams. ‘underrepresented minority’ considered harmful, racist language. *Communications of the ACM*, 2020.
- [429] Anne-Marie Willis. Ontological designing. *Design philosophy papers*, 4(2):69–92, 2006.

- [430] Jeannette M. Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006.
- [431] Rupert Wingfield-Hayes. The beauty contest winner making japan look at itself. *BBC News Tokyo*, June 2015.
- [432] Langdon Winner. Do artifacts have politics? *Daedalus*, pages 121–136, 1980.
- [433] Terry Winograd, Fernando Flores, and Fernando F. Flores. *Understanding computers and cognition: A new foundation for design*. Intellect Books, 1986.
- [434] Patrick Wolfe. *Traces of history: Elementary structures of race*. Verso Books, 2016.
- [435] Steve Woolgar. The turn to technology in social studies of science. *Science, Technology, & Human Values*, 16(1):20–50, 1991.
- [436] Yifan Wu, Joseph M. Hellerstein, and Arvind Satyanarayan. B2: Bridging code and interactive visualization in computational notebooks. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, pages 152–165, New York, NY, USA, 2020. Association for Computing Machinery.
- [437] Malcolm X. *Malcolm X speaks: Selected speeches and statements*. Grove Press, 1990.
- [438] Qian Yang, Aaron Steinfeld, Carolyn Rosé, and John Zimmerman. Re-examining whether, why, and how human-ai interaction is uniquely difficult to design. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–13, New York, NY, USA, 2020. Association for Computing Machinery.
- [439] Kyra Yee, Uthaiapon Tantipongpipat, and Shubhanshu Mishra. Image cropping on twitter: Fairness metrics, their limitations, and the importance of representation, design, and agency. *arXiv preprint arXiv:2105.08667*, 2021.
- [440] Renzhe Yu, Hansol Lee, and René F Kizilcec. Should college dropout prediction models include protected attributes? In *Proceedings of the Eighth ACM Conference on Learning@ Scale*, pages 91–100, 2021.

- [441] Hans Yuan and Yingjun Cao. Hybrid pair programming - a promising alternative to standard pair programming. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education, SIGCSE '19*, page 1046–1052, New York, NY, USA, 2019. Association for Computing Machinery.
- [442] Alexei Yurchak. *Everything was forever, until it was no more: The last Soviet generation*. Princeton University Press, 2013.
- [443] Anna Zeng and Will Crichton. Identifying barriers to adoption for rust through online discourse. *arXiv preprint arXiv:1901.01001*, 2019.
- [444] Lei Zhang and Steve Oney. Flowmatic: An immersive authoring tool for creating interactive scenes in virtual reality. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, pages 342–353, New York, NY, USA, 2020. Association for Computing Machinery.
- [445] Zara Zimbaro. Cultural politics of humor in (de) normalizing islamophobic stereotypes. *Islamophobia Studies Journal*, 2(1):59–81, 2014.
- [446] John Zimmerman, Jodi Forlizzi, and Shelley Evenson. Research through design as a method for interaction design research in hci. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '07*, pages 493–502, New York, NY, USA, 2007. Association for Computing Machinery.
- [447] Tukufu Zuberi, Eduardo Bonilla-Silva, et al. *White logic, white methods: Racism and methodology*. Rowman & Littlefield Publishers, 2008.
- [448] Konrad Zuse. Der Plankalkül, 1945. Transcribed report. Courtesy of Konrad Zuse Internet Archive (<http://zuse.zib.de>).
- [449] Konrad Zuse. Urschrift des Plankalküls, 1945. Original notebook (scanned). Courtesy of Konrad Zuse Internet Archive (<http://zuse.zib.de>).
- [450] Konrad Zuse. *The computer-my life*. Springer Science & Business Media, 1993.