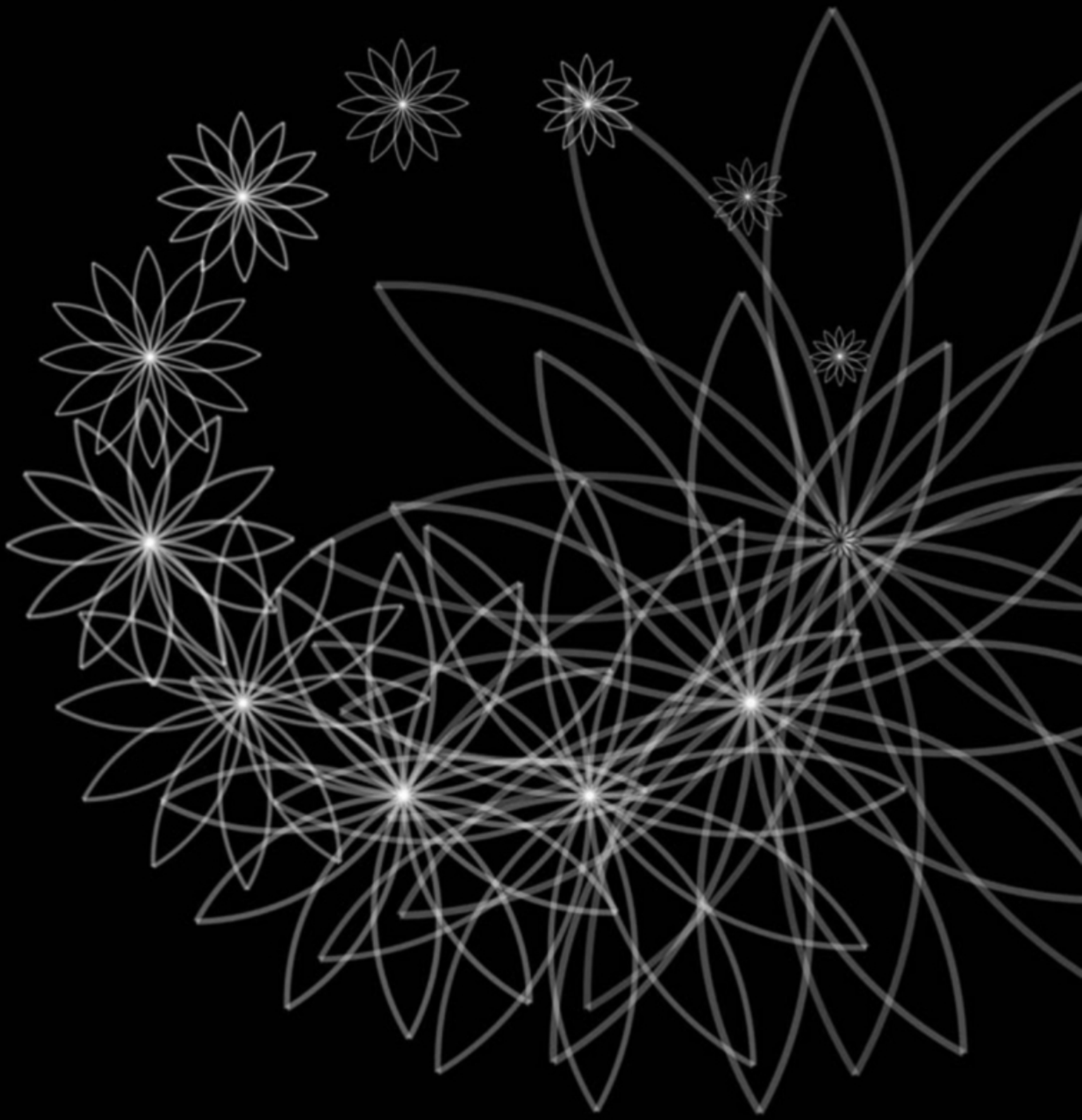
The Software Arts

Warren Sack



The Software Arts

Warren Sack

The MIT Press
Cambridge, Massachusetts
London, England

This book was set in ITC Stone Serif Std Medium by Westchester Publishing Services. Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Names: Sack, Warren, author.
Title: The software arts / Warren Sack.
Description: Cambridge, MA : The MIT Press, [2019] | Series: Software studies |
Includes bibliographical references and index.
Identifiers: LCCN 2018032131 | ISBN 9780262039703 (hardcover : alk. paper)
Subjects: LCSH: Electronic data processing—Popular works. | Computer software—
Popular works.
Classification: LCC QA76 .S2164 2019 | DDC 005.3—dc23
LC record available at https://lccn.loc.gov/2018032131

10 9 8 7 6 5 4 3 2 1

Contents

Series Foreword ix
Foreword by John Rajchman xi
Acknowledgments xvii

1 **Introduction** 1
2 **Translation** 31
3 **Language** 57
4 **Algorithm** 79
5 **Logic** 107
6 **Rhetoric** 145
7 **Grammar** 203
8 **Conclusion** 243

Notes 261
Bibliography 327
Index 357

Series Foreword

Software is deeply woven into contemporary life—economically, culturally, creatively, politically—in manners both obvious and nearly invisible. Yet while much is written about how software is used, and the activities that it supports and shapes, thinking about software itself has remained largely technical for much of its history. Increasingly, however, artists, scientists, engineers, hackers, designers, and scholars in the humanities and social sciences are finding that for the questions they face, and the things they need to build, an expanded understanding of software is necessary. For such understanding they can call upon a strand of texts in the history of computing and new media, they can take part in the rich implicit culture of software, and they also can take part in the development of an emerging, fundamentally transdisciplinary computational literacy. These provide the foundation for Software Studies.

Software Studies uses and develops cultural, theoretical, and practice-oriented approaches to make critical, historical, and experimental accounts of (and interventions via) the objects and processes of software. The field engages and contributes to the research of computer scientists, the work of software designers and engineers, and the creations of software artists. It tracks how software is substantially integrated into the processes of contemporary culture and society, reformulating processes, ideas, institutions, and cultural objects around their closeness to algorithmic and formal description and action. Software Studies proposes histories of computational cultures and works with the intellectual resources of computing to develop reflexive thinking about its entanglements and possibilities. It does this both in the scholarly modes of the humanities and social sciences and in the software creation/research modes of computer science, the arts, and design.

The Software Studies book series, published by the MIT Press, aims to publish the best new work in a critical and experimental field that is at once culturally and technically literate, reflecting the reality of today's software culture.

Foreword: Software as a Mode of Thinking—An Introduction

John Rajchman

The modern digital computer is the invention of two distinguished mathematicians, Alan Turing and John von Neumann, working in the heyday of a rich debate about numbers and logic and a grand search for the "laws of thought," which they then tried to introduce—perhaps it would be better to say "translate"—into the workings of a new kind of machine, the computer.

The story of this invention has often been told, that of Bletchy Park and The Institute for Advanced Study. Born of the urgencies of war, often elaborated in secrecy in government facilities against a formidable foe, and mobilizing its own science-tech sector, the invention would assume new forms after the war. It would become part of an ever-expanding "military industrial complex," with us now as much as ever, with our giant global Internet companies, surveillance, hacktivism, cybersecurity, smart cities, and infrastructures. In the process, "platforms" themselves would pass from mainframe to PC to smartphone, increasing in speed, efficiency, and reach, and leading to the operations of our great number-crunching algorithms in finance, politics, and social media. The invention of the computer by these two great mathematicians, carried on in military secrecy, in short, has led to an enormous complex in government and economics alike, touching on many aspects of the ways we think and live.

But what role did actual programming play in this history? How did the "translations" of various activities into this complex itself evolve, assuming new forms and functions? What role might programming yet play in the matter of "digital intelligence" today? Such is the complex problem this new study of the origins and nature of software sets out to raise and, in the first place, to formulate. How can we do the history of software itself: What exactly is it? How should we study it? In particular, what ever happened to the great logicist dream of discovering "laws of thought," which inspired Turing and von Neumann, extended at the time in striking ways by Gödel and Hilbert? In what ways did this logicist background help foster a picture of thinking or

"smartness" itself, conceived as a matter of following rules in a finite number of steps, independent of human interaction of any kind—a picture still very much alive today in digital culture, popular as well as more sophisticated? What would it mean to see programming instead, from the start, as belonging to a different, more materially rooted history—more like cooking up ways of doing things, inventing "recipes" for integrating our smart machines into the larger demands of politics, war, finance, commerce, and everyday life? What role in particular might the arts play in this expanded history? What would it mean to see the "digitalization" of artistic media as part of it? Could we adapt what Bruno Latour calls "translation" for these purposes, challenging the idea, already found with "the computer," that digital intelligence arises by simply creating artificial forms for human activities? How then does such transformative translation work in the case of software? In what ways do the translations that software helps bring about introduce something new, for which no human model previously existed, challenging our very ideas of natural or nonartificial human activity? What role, in short, has software played in the ways we talk about and see things—and therefore act?

In fact, even to talk about the translations effectuated by computer programming, we often lack a preexisting vocabulary. The very words we are accustomed to using—"computer," "program," even "algorithm"—of necessity draw on predigital languages and practices, in genealogies we can now retrospectively examine. We see this, for example, in the case of "media" and "media studies" of the sort exemplified by Lev Manovich, part of the larger disciplinary framework through which we talk about and see digitalization. The terms "media" and "medium" were in fact drawn from the arts and journalism, then television, areas that themselves are being transformed by the rise of smart machines. In the arts today, for example, we see a movement away from the very ideas of media and medium, still important for a group like Radical Software, in favor of something more like "ecology of images." How might we start to chart these developments, studying software as a complex, evolving mode of thinking, within divisions of knowledge and artistic practices? Raising all these questions at once, this patient study, six years in the making, becomes a search for method and a plea for new ways of thinking, and the role that software practice might yet play in them.

What, then, is software as a mode of thinking? *The Software Arts* exposes an apparent paradox. While the grand logicist dream that led to the invention of the computer has long lost its philosophical hold on us, in many ways its ghost lives on in digital culture, in the very idea of smartness it helped introduce, in ways Warren Sack starts to trace: Turing machines, artificial intelligence, "cognitivism." He finds one turning point in Noam Chomsky, whose search for innate syntactic structures in language would lead to attempts to relocate such logic in the brain, or universal "neural cognition." To see software instead as a mode of thinking is to reverse the question—not whether our

This new space of exchange and analysis across the arts and sciences is one that must itself be invented. That is the suggestion that emerges from this study. It is not simply a matter of getting the two "communities," intellectual-artistic and military-commercial, to talk more with each other, but rather to encourage the creation of a new space of discussion, for the problem of digital intelligence today is not simply one of the two cultures, scientific and humanistic. (It is not clear that the "intelligence" of software nerds is very mathematical at all; it is in fact drawn from many other sources.) The question it poses is not in the first place a matter of machines and us, artifice and nature, regarding who is in control. (The history of software is a history of modes of thinking that are at once artificial and natural, scientific and artistic.) The problem of digital intelligence is rather how to invent new ways of working together, outside the confines within which our thinking is now kept, and for that no simple return to a bookish "humanities" will suffice. The workings of our smart machines need to be analyzed at once in the arts and the sciences in ways and through methods and means that help create new relations between them. Only then will the reigning idea of smartness be replaced by something more like a kind of collective intelligence, working and thinking together across many domains and disciplines. The force and originality of this study is to show how software—software as a mode of thinking—has a key role to play in this process.

Acknowledgments

This book has been written in Santa Cruz, California, and Paris, France. At the University of California, Santa Cruz (UCSC), I would like to thank the faculty, staff, and students in the Film+Digital Media Department, the Digital Arts & New Media MFA Program, the History of Art and Visual Culture Department, the History of Consciousness Department, the Computational Media Department, the Center for Cultural Studies, the Science and Justice Research Center, and the Center for Games and Playable Media. I have also benefited from logistical and financial support offered to me for this book by the staff and the Dean of the Arts.

The American Council of Learned Societies granted me an ACLS Digital Innovation Fellowship that allowed me to begin writing this book in Paris, hosted by Bruno Latour at the Médialab of Sciences Po and by Françoise Detienne at the Département Sciences Economiques et Sociales of Télécom ParisTech. I am deeply grateful for financial support from the ACLS and for the collegial and institutional support Bruno and Françoise and their respective networks offered me that year.

Also in Paris, I offer my thanks to Bernard Stiegler and Vincent Puig of the Institut de recherche et de l'innovation (IRI) at the Centre Georges Pompidou, where I have been invited to speak to and learn from them and our colleagues of the Digital Studies network. I am equally grateful to Samuel Bianchini and Emmanuel Mahé at EnsadLab of l'École des Arts Décoratifs (EnsAD) for the many interactions I have enjoyed with them and their doctoral students.

Most recently, my institutional home in Paris has been at the Paris Institute for Advanced Study (IAS), where this book and I benefited from a fellowship with the financial support of the French State program Investissements d'avenir, managed by the Agence Nationale de la Recherche (ANR-11-LABX-0027-01 Labex RFIEA+). I was a fellow at the Paris IAS during the fall of 2015 and the fall of 2016, housed in the spectacular Hôtel de Lauzun. Director Gretty Mirdal has created an extraordinarily productive interdisciplinary exchange. Simon Luck, the scientific coordinator, kept me

connected to many research communities in and around Paris. Librarian Geneviève Marmin helped me navigate the libraries and archives of the ENS, MSH, the Sorbonne, Jussieu, and the CNAM.

Over the course of my two stays at the Paris IAS, I got to know forty other fellows. Each fellow made a contribution to my writing and thinking through our weekly colloquia and our daily lunches. I would like to thank Keith Baker, Sean Takats, and Charles Walton for providing me with suggestions about what to read and what to consider as my research began to move more deeply into the specifics of the eighteenth-century *Encyclopédie* of Diderot and d'Alembert. When I was working on the chapter on algorithms, Carlos Gonçalves generously offered his expert insights into ancient Mesopotamian mathematics. And I was very influenced by Nachum Dershowitz's approach to his research, in which he integrates both historical and technical details of computer science. Nachum and I also co-produced a workshop with our Parisian colleagues on the topic of software and the digital humanities, and we both contributed to a celebration of Ada Lovelace organized by Director Mirdal. For my IAS colloquium presentations, I was honored to have my Parisian colleagues Patrice Maniglier, Bernard Stiegler, and Jean-Gabriel Ganascia as my respondents.

This book has benefited from several other workshops in which my friends and colleagues reviewed the draft manuscript.

In March of 2013, Matthew Fuller and Noortje Marres hosted a workshop for me at Goldsmiths College, University of London. I would like to thank them, Olga Goriunova, Nina Wakeford, and the then-doctoral students—Beatrice Fazi, Ana Gross, Rosa Menkman, and David Moats—for the rigorous yet generous readings they gave to the manuscript.

In June of 2013, I hosted an ACLS-sponsored workshop in Paris that brought together colleagues from the Médialab, Télécom, and other research centers. Participants included Michael Baker, Audrey Baneyx, Valérie Beaudouin, Samuel Bianchini, Dominique Cunin, Jérôme Denis, Françoise Detienne, Dana Dimenescu, Paul Edwards, Annie Gentes, Marie Gil, Paul Girard, Jennifer González, Mathieu Jacomy, Benjamin Loveluck, Patrice Maniglier, Dominique Pasquier, Jean-Christophe Plantin, Serge Proulx, Vincent Puig, Everardo Reyes-García, and Tommaso Venturini.

In July of 2013, Richard Rogers of the University of Amsterdam invited me to speak at the Digital Methods Summer School about what, ultimately, became chapter 6 of this book, on the topic of rhetoric.

The ACLS fellowship allowed me to carry over funds from my fellowship to hold a workshop at UCSC in June of 2014. I wish to thank the following participants: Sophie Bargues-Rollins, David Bates, Jon Beller, Alan Christy, Chris Connery, Joe Dumit, Shelly Errington, Carla Freccero, Elaine Gan, Jennifer González, John Kadvany, Deirdra

"Squinky" Kiai, Dilan Mahendran, Michael Mateas, Soraya Murray, Abram "Aphid" Stern, Mike Travers, Lyle Troxell, and Noah Wardrip-Fruin. Now, listening to the recordings of our discussions, I am impressed with how our free-ranging discussion ultimately gave the book a much sharper focus.

During my last couple of stays in Paris, Patrice Maniglier included me in the "Sémiomaths" (semiology of mathematics) working group he co-organizes with Juan Luis Gastaldi and David Rabouin. I thank them for letting me join some of their meetings and for devoting a meeting in 2015 and another in 2018 to a discussion of my book manuscript.

I am deeply grateful to Doug Sery, my editor at the MIT Press, who has encouraged and supported this project since 2012. I thank the external reviewers for detailed and insightful analyses of the manuscript. When revising the text, I also had the good fortune to work with the remarkable developmental editor Kathryn Chetkovich, whose clarity and thoughtfulness significantly improved the final result.

My motivation for writing this book has been vitally connected to my role as a teacher. Hundreds of undergraduate students in my recent offerings of a large lecture course at UCSC, Introduction to Digital Media, have wrangled with earlier versions of this text. It has also benefited from the readings and comments of my current and former PhD and MFA students who have been teaching assistants for the undergraduate course; taken some version of my doctoral seminar, Software Studies; or my graduate course, Introduction to Programming for the Arts; or have decided to take a qualifying examination with me in software studies; or conducted research with me on Open Source Software development. I would especially like to acknowledge the following former and current graduate students for their insights and support over the years: Nicolas Ducheneaut, Elaine Gan, Fabiola Hanna, Nik Hanselmann, Meredith Hoy, Chris Kerich, Nick Lally, Dylan Lederle-Ensign, Dilan Mahendran, Michael McCarrin, Karl Mendonca, Abram "Aphid" Stern, and Lindsay Weinberg.

Many of the ideas in this book had their beginnings years ago when I was a member of the feminist studies of science and technology (STS) reading group that met in the basement of the Yale Computer Science Department in the 1980s; a Chateaubriand Fellow at the Department of Computer Science of the University of Paris 8 (St. Denis); an unofficial member of the STS Research Cluster of the Center for Cultural Studies at UCSC that was organized by Donna Haraway's graduate students in the late 1980s and early 1990s; a co-organizer of the Narrative Intelligence Reading Group that met in the basement of the MIT Media Lab in the 1990s; a visitor at the Center on Organizational Innovation at Columbia University; and a visitor at the Centre de sociologie de l'innovation at Mines ParisTech. I would like to thank past and present members of these inspiring communities.

initial professionalization of science and engineering, it becomes clear that computing grew out of the arts.

The Software Arts is also a reading of the texts of computing—code, algorithms, and technical papers—that emphasizes continuities between prose and programs.[3] Historically, it is possible to say that this position was first sketched out in the seventeenth century in proposals to develop artificial, philosophical languages that were used to knit together the liberal arts (e.g., logic, grammar, and rhetoric, the liberal arts of language) and the mechanical arts (e.g., those practiced by artisans in workshops producing pins, stockings, locks, guns, and jewelry).[4] In brief, these artificial languages became what we know today as computer programming languages. The claim is that contemporary, artificial languages have shaped and been shaped by the arts and have rearticulated the relationship between the liberal arts and the mechanical arts—an assembly we currently call art, design, the humanities, and technology.

Programming languages are the offspring of an effort to describe the mechanical arts in the languages of the liberal arts. Writing software is a practice of writing akin to the activity of novelists, playwrights, screenwriters, speechwriters, essayists, and academics in the arts and the humanities. Consequently, contemporary education, research, industry, and technology development all need to change to better recognize how the arts sit at the center of computing.

Apple's Artists

In 1995, Apple cofounder Steve Jobs said, "Part of what made the Macintosh great was that the people working on it were musicians, poets and artists and zoologists and historians who also happened to be the best computer scientists in the world.... And they brought with them, we all brought to this effort, a very liberal arts attitude."[5] Long after the introduction of the original Macintosh computer, Jobs was still describing the liberal arts as an Apple competitive advantage. At the launch of a new model of the iPad tablet computer, Jobs said, "It is in Apple's DNA that technology alone is not enough—it's technology married with liberal arts, married with the humanities, that yields us the results that make our heart sing."[6] In this book, I will argue that Jobs was right: the arts and the humanities are at the heart of computing.[7]

In the United States, Jobs's comments are remarkable today, when the arts and humanities are under siege with demands that students receive preprofessional training instead of a fine arts or liberal arts education.[8] The increasing disregard for a liberal arts education is misguided.[9] If Jobs was right, education needs to change. Computing

education needs to be redesigned to recognize its rightful place in the liberal arts, and the humanities disciplines of the contemporary liberal arts need to be extended to acknowledge their position at the heart of the computer revolution.[10]

If Jobs was right, it also becomes possible to imagine how computing research and development can be pursued as forms of arts research and humanities scholarship. With this insight, the path to the next "insanely great"[11] computer technology widens to become a great expressway accommodating a much larger and more diverse group of fellow travelers.

To emphasize the centrality of the arts would almost certainly help the computer industry with its long-standing diversity problems. At least that is the thinking that drove the summer 2014 diversity campaign in which Apple described itself this way: "From the very beginning, we have been a collective of individuals. Different kinds of people from different kinds of places. Artists, designers, engineers and scientists, thinkers and dreamers. An intersection of technology and the liberal arts. Diverse backgrounds, all working together."[12]

Computing and the Arts

Beyond Steve Jobs and Apple are a number of important computer scientists who have also put the arts at the center of computing. For example, Harold Abelson, Gerald Sussman, and Julie Sussman wrote a programming textbook for their undergraduate students at the Massachusetts Institute of Technology. Their textbook, *The Structure and Interpretation of Computer Programs*, embodies this alternative vision of computing. The authors state in their introduction:

> Underlying our approach to this subject is our conviction that "computer science" is not a science and that its significance has little to do with computers. The computer revolution is a revolution in the way we think and in the way we express what we think. The essence of this change is the emergence of what might best be called procedural epistemology—the study of the structure of knowledge from an imperative point of view, as opposed to the more declarative point of view taken by classical mathematical subjects. Mathematics provides a framework for dealing precisely with notions of "what is." Computation provides a framework for dealing precisely with notions of "how to."[13]

By distinguishing classical mathematics from computation—"what is" as distinguished from "how to"—the authors articulate a position of "procedural epistemology," but rather than coining the new phrase "procedural epistemology," they could simply have said that computing is an art. "Art" in its original sense means how-to knowledge—as used in phrases such as "martial arts" and "arts and crafts."

In their book, Abelson, Sussman, and Sussman emphasize one aspect of epistemology: that computing constitutes a new way of thinking. Computer scientist Edsger Dijkstra stated the case like this: "[Computers] have had a great impact on our society in their capacity of tools, but in that capacity their influence will be but a ripple on the surface of our culture, compared with the much more profound influence they will have in their capacity of intellectual challenge without precedent in the cultural history of mankind."[14]

This form of research and education, with a focus on the implications for cultural history, has been pursued with a mixture of methods that weave together ideas from the arts, the humanities, and mathematics. Donald Knuth, Professor Emeritus of the Art of Computer Programming at Stanford University, advocates a method he calls "literate programming": "Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.... The practitioner of literate programming can be regarded as an essayist, whose main concern is with exposition and excellence of style."[15]

Knuth sees programming as an art and as literature. Practitioners of "literate programming" and "procedural epistemology" are essayists, writers, and expositors. For Steve Jobs, Donald Knuth, Edsger Dijkstra, Harold Abelson, Gerald Sussman, and many other important computer scientists in the world (e.g., many who have won the Turing Award, the analog of the Nobel Prize for computer science), computing is part and parcel with the liberal arts. By arguing that the arts are at the heart of computing, I am arguing neither a radical nor a marginal point.

Computing and Engineering

Unfortunately, even though this argument has been made repeatedly and with great authority, it remains institutionally marginalized and generally unpopular. Institutionally—in both education and industry—the winning arguments have positioned computing either within the sciences or as a form of engineering. As a result, in universities, most computing departments are positioned in schools of science or engineering and away from schools of the arts and humanities.

While these "winning" arguments have been reified in the shaping of institutions, if we look closely at the arguments as originally stated and as pursued to date, we see that they are based on undefined terms and nonobvious and unstable analogies between computing and the subjects and objects studied and produced by science and engineering disciplines. Casting computing in a new disciplinary mold is frequently,

at least initially, not commonsensical. For example, in a remark at the first Software Engineering Conference, convened in 1968 by the North Atlantic Treaty Organization (NATO) Science Committee, an analogy was drawn between software production and engineering: "The phrase 'software engineering' was deliberately chosen as being provocative, in implying the need for software manufacture to be based on the types of theoretical foundations and practical disciplines that are traditional in the established branches of engineering."[16]

In an article on the history of software engineering, Michael Mahoney writes the following about this opening gambit of the conference:

> The phrase was indeed provocative, if only because it left all the crucial terms undefined. What does it mean to "manufacture" software? Is that a goal or current practice? What, precisely, are the "theoretical foundations and practical disciplines" that underpin the "established branches of engineering"? What roles did they play in the formation of the engineering disciplines? Is the story the same in each case? The reference to "traditional" makes the answer to that question a matter of history—analyzing how the fields of engineering took their present form and searching for historical precedents, or what we have come to refer to as "roots."[17]

As Mahoney goes on to show in this article and subsequent scholarship, these terms remain undefined, decades after this first conference.[18]

One of the participants at the first Software Engineering Conference of 1968, Alexander D'Agapeyeff, had the following complaint: "Programming is still too much of an artistic endeavor."[19] The presupposition inherent in this complaint is, of course, that programming is already an art but that the aspiring software engineers would like it to be otherwise.

Indeed, "software engineering" remains an unrealized goal despite its current institutional success. Software engineers would like their discipline to be accepted as a form of engineering, but they are repeatedly unsure that it is. As Mahoney wrote in the introduction to his article, "It is … not hard to find doubts about whether its current practice meets those criteria and, indeed, whether it is an engineering discipline at all.… [It has been declared that] 'Software engineering is not yet a true engineering discipline, but it has the potential to become one.' From the outset, software engineering conferences have routinely begun with a keynote address that asks, 'Are we there yet?'"[20]

Is Computing a Science?

In 1967, a year before the first Software Engineering Conference, Allen Newell, Alan Perlis, and Herbert Simon published a letter in the journal *Science* arguing that computing is not (just) engineering but is also a science: "Professors of computer science are

often asked: 'Is there such a thing as computer science, and if there is, what is it?' The questions have a simple answer: Wherever there are phenomena, there can be a science to describe and explain those phenomena. Thus, the simplest (and correct) answer to 'What is botany?' is, 'Botany is the study of plants.' And zoology is the study of animals, astronomy the study of stars, and so on. Phenomena breed sciences. There are computers. Ergo, computer science is the study of computers."[21]

This letter had tremendous persuasive power and, arguably, launched the founding of many university computer science departments. It incorporates a number of tropes—rhetorical techniques—to press the case. Consider, for instance, the opening statement, "Professors of computer science are often asked...." In 1967, there were very few professors of computer science, because the first university computer science department had been founded only five years earlier, in 1962, at Purdue University.[22] All three of the letter writers were affiliated with Carnegie Mellon University (then called Carnegie Institute of Technology), where one of the first computer science departments in the country, after Purdue's, was founded in 1965 by the letter writers. Indeed, Alan Perlis had moved to Carnegie Tech from Purdue and was the first head of its Computer Science Department.[23] So, at the time, their opening line would have been akin to three of the first astronauts writing "astronauts are often asked...."

While this letter may have been the first word on the topic, it was hardly the last. In their 1976 Turing Award lecture, "Computer Science as Empirical Inquiry," Allen Newell and Herbert Simon group together geology, astronomy, and economics as "empirical disciplines" and then compare them as a group to computer science.[24] But geology, astronomy, and economics are not similar in any obvious way. Moreover, their intersection certainly does not provide a clear set comparable to an emerging fourth discipline, like computer science.

Newell and Simon elaborate on these unlikely comparisons with further far-fetched analogies: "Each program that is built is an experiment. It poses a question to nature, and its behavior offers clues to an answer."[25] Unaddressed by Newell and Simon is the poetic license they employ to anthropomorphize "nature" (a term they capitalize later in the article) into a being that can answer questions; the metaphorical notion that textual productions, like programs, are "built" rather than written; and the copular statement that makes building into a form of experimenting. That these kinds of loose and fanciful analogies carried the day and convinced many that the study of computing is a science may be puzzling but was—and, undeniably, still is—a winning rhetoric: "computer science" is more than analogical apposition; it is a large, growing, and well-funded discipline.

Simon tried to write the definitive word on this by publishing a book on the topic, *Sciences of the Artificial* (with three editions, in 1969, 1981, and 1996), but he never

If numbers were "native" to computers, no such computer science specialty would be necessary.[40]

My point is that although numbers and operations on numbers—such as arithmetic—can be approximated with a computer, computers are not numerical machines. They are language machines, and numbers are just a very common domain of application. Imagining computers only as powerful calculators confuses the machine itself with a single important application of computer technology.

Let me underline this pitfall with an absurd example. If I use the microwave oven in my kitchen mostly as a means to make popcorn, does that mean that the microwave is essentially a "popcorn machine"?

My point is controversial because it is, of course, a counternarrative to the most commonly told history, in which computers are figured as information technologies and are thus tied to information, quantification, and mathematics. In contrast, in the story I want to tell, computers are a coupling of the liberal arts and the mechanical arts—what today we would call the knowledge of artisans, artists, humanists, and designers. In the first story, computers are the materialization of mathematics and science. In the second story, computers are the manifestation of methods and theories from the arts and humanities.

Computing and the Liberal Arts

If computing can be conceived of as something other than just science or just engineering, what are the alternatives? Ironically, or perhaps characteristically, Alan Perlis—who, as mentioned earlier, argued in 1967 that computing *is* a science and, as a participant at the first Conference of Software Engineering in 1968, helped to articulate a vision of software production as engineering—described, in 1962, how computer programming should be integrated into a liberal arts education.[41] This third approach, computing as an intrinsic part of a liberal arts education, also advanced by Perlis, has more recently been revived by, for example, computational media theorists and practitioners Michael Mateas[42] and Ian Bogost in their respective advocacies for a "procedural literacy." Bogost specifically turns to an examination of the teaching of the language arts, the trivium, in his exploration of what a procedural literacy could mean for education and learning.[43]

Perlis's third path, the idea that computing research and education can be pursued within the liberal arts, is comfortable for computer scientists who are programming language designers, since they find it quite natural to imagine that computing is primarily about the creation and use of language. Alan Perlis was one of the designers of the ALGOL language.[44] Gerald Sussman, cited earlier, was co-designer of the Scheme

programming language.[45] Casey Reas and Ben Fry designed Processing, a programming language to help artists and designers learn how to program.[46] Those who have been especially articulate about computing-as-language-art have included the designers of programming languages produced to teach children and novices to program. Alan Kay co-invented—originally for children—the object-oriented programming language Smalltalk.[47] Mitchel Resnick's Scratch programming language has transformed computing education for young children.[48] Resnick's mentor, Seymour Papert, was the co-designer of a programming language for children called LOGO.[49] Papert was pivotal in tearing down the walls between the language arts, science, and mathematics. In his book *Mindstorms,* Papert put it like this: "Plato wrote over his door, 'Let only geometers enter.' Times have changed. Most who now seek to enter Plato's intellectual world neither know mathematics nor sense the least contradiction in their disregard for his injunction. Our culture's schizophrenic split between 'humanities' and 'science' supports their sense of security. Plato was a philosopher, and a philosopher belongs to the humanities as surely as mathematics belongs to the sciences."[50]

Papert's comments make us recall that the traditional liberal arts, as studied and practiced in the early modern era of Europe, included both the trivium (the arts of language) and the quadrivium (the arts of number). At that time, there was no strict boundary to be drawn between what today we call the arts and the humanities and the sciences.

A Short History of the Liberal Arts

What might a recasting of computing as part of the liberal arts look like? A short history of the liberal arts will show that they have been expanding and diversifying for centuries and that the design of programming languages, the languages of software, are the latest version of a very old dream of the liberal arts—to find just the right words, just the right language.

What are the liberal arts? The *Oxford English Dictionary* (*OED*) defines the liberal arts as:

> Originally: the seven subjects of the trivium (grammar, rhetoric, and logic) and quadrivium (arithmetic, geometry, music, and astronomy) considered collectively (now historical).

American Catholic nun Miriam Joseph wrote a widely read college textbook in which she defined the seven liberal arts as follows:

> The *trivium* includes those aspects of the liberal arts that pertain to mind, and the *quadrivium,* those aspects of the liberal arts that pertain to matter. Logic, grammar, and rhetoric constitute

> the *trivium*; and arithmetic, music, geometry, and astronomy constitute the *quadrivium*. Logic is the art of thinking; grammar, the art of inventing symbols and combining them to express thought; and rhetoric, the art of communicating thought from one mind to another, the adaption of language to circumstance. Arithmetic, the theory of number, and music, an application of the theory of number (the measurement of discrete quantities in motion), are the arts of discrete quantity or number. Geometry, the theory of space, and astronomy, an application of the theory of space, are the arts of continuous quantity or extension.... These arts of reading, writing, and reckoning have formed the traditional basis of liberal education, each constituting a field of knowledge and the technique to acquire that knowledge. The degree bachelor of arts is awarded to those who demonstrate the requisite proficiency in these arts, and the degree master of arts, to those who have demonstrated a greater proficiency.[51]

Many others summarize the trivium as the arts of language and the quadrivium as the arts of number.

Media scholar Marshall McLuhan wrote his dissertation on a history of three of the liberal arts, specifically the trivium.[52] He emphasized the historical centrality and continuity of the liberal arts, citing A. F. Leach: "It is hardly an exaggeration to say that the subjects and the methods of education remained the same from the days of Quintilian to the days of Arnold, from the first century to the mid-nineteenth century of the Christian era."[53] Arguably, in much of Europe, the liberal arts were at the heart of education for a millennium. Yet, as McLuhan's history makes clear, the static picture of the liberal arts projected by Sister Joseph was only a snapshot of a specific historical period. The definition of the arts and their relationships to each other changed dramatically from one era to the next, and thus "grammar," "logic," and "rhetoric" today are not necessarily the same as their historical precedents.

In the United States, many elite institutions are still called liberal arts colleges. Yet, what is called a "liberal arts education" today is no longer the Aristotelian endeavor outlined by Sister Joseph in 1948; nor is it exactly the silhouette A. F. Leach saw in 1911. In early modern Europe, the liberal arts were distinguished from mechanical or manual arts.[54] Today, in some countries, such as the United States, liberal arts colleges do include the fine arts. Elsewhere, however—for example, in France and Germany—art colleges and technical schools are separate from the university. But A. F. Leach's choice of the mid-nineteenth century as a critical moment for the transformation and expansion of the liberal arts throughout Europe and the United States is compelling, because it was then that both industrialization and the rise of the humanities changed the liberal arts by integrating them with the mechanical arts.

The contemporary definition of the liberal arts puts them in opposition to science and technology. I elided from the *OED* citation earlier this crucial sentence: "In later use more generally: arts subjects as opposed to science and technology (now chiefly

North American)." What happened in nineteenth-century American education that seems to have made technology and the liberal arts antonyms but at the same time paradoxically expanded a liberal arts education to include the mechanical arts, engineering, and technology?

In 1936, Henry Seidel Canby wrote a memoir set in the American college, specifically centered on his experience at Yale College. In *Alma Mater: The Gothic Age of the American College*,[55] Canby pointed out that the college had been radically transformed between 1870 and 1910. Industrialization, the rise of the corporation, and the invention of the American research university all played a part in changing college.

Before 1870, college was for an elite few. In the United States of 1870, there were about 50,000 undergraduates.[56] By 1920, there were an order of magnitude more undergraduates, over half a million. Before 1870, undergraduates enrolled in college were primarily pursuing a liberal arts education prior to entry into one of three specialties: divinity, law, or medicine. But during this period of industrialization, new, specialized forms of knowledge were developed to deal with the introduction of a myriad of emerging machines and new forms of production and distribution. Thus, for many, after 1870 a college education was no longer synonymous with a traditional liberal arts education.

The first federal aid for higher education in the United States was the 1862 Morrill Land Grant College Act: "An Act Donating public lands to the several States [and Territories] which may provide colleges for the benefit of agriculture and the Mechanic arts…in order to promote the liberal and practical education of the industrial classes in the several pursuits and professions in life."[57] The Morrill Act provided the founding financial support for many of the great public (and some private) universities of the United States, including the University of California, Berkeley; Purdue (later the site of the first computer science department); University of Wisconsin–Madison; University of Maryland, College Park; Massachusetts Institute of Technology; and Cornell. At the time, it was argued that the industrial era required a new form of college education incorporating "utilitarian" and "democratic" forms of knowledge.[58] Where previously a select group of gentlemen were instructed in the liberal arts before starting careers as clergymen, lawyers, or medical doctors, the Morrill Act signaled that a larger population needed to be educated in some hybrid of the mechanical and the liberal arts so that they might become accountants, engineers, and technical experts of the many, increasingly specialized, disciplines important for industrial capitalism. These new universities created by the Morrill Act were minted in a very different die than the older, originally ecclesiastical, colleges of early America, such as Harvard, Yale, and Princeton.

This thread of change—the expansion of college education to concern new forms of technical expertise[59]—was intertwined with another: the rise of the research university

in the nineteenth century.[60] Prior to the nineteenth century, universities were primarily teaching institutions where professors were paid to teach an established canon of knowledge, not to develop new forms of knowledge and technology. In the nineteenth century, the research university was invented in Germany, especially Prussia. This new model of the university was imported to the United States by then-new universities, such as Johns Hopkins and the University of Chicago, and, after that, was adapted by long-standing institutions, such as Yale. Central to this new model of the university—and a departure from the older models—was the emphasis on empirical scientific research; the change in the duties of professors to pursue research in addition to teaching; and the increasing investment in secular forms of knowledge.[61]

As McLuhan demonstrates in his history of the trivium—the three liberal arts devoted to language—a liberal arts education was, for centuries, tantamount to a Christian religious education. Nevertheless, since this form of education properly started in ancient Greece, even during the early modern period it combined ancient Greek philosophy (especially Aristotelian philosophy) with teachings of the Catholic Church (especially those of Thomas Aquinas). So, the liberal arts have always been interdisciplinary.

The emergence of the humanities from the liberal arts arose from a secularization of this body of knowledge. This third strand of change—secularization—was woven with the influence that the increasing ubiquity of technologies of industrialization and the rise of the research university had on education. Secularization displaced the sacred and, in its stead, centralized the study of the human.

In the introduction to their text *Digital_Humanities*, Anne Burdick, Johanna Drucker, Peter Lunenfeld, Todd Presner, and Jeffrey Schnapp encapsulate this history in one paragraph:

> While the foundations of humanistic inquiry and the liberal arts can be traced back in the West to the medieval *trivium* and *quadrivium*, the modern human sciences are rooted in the Renaissance shift from a medieval, church-dominated, theocratic worldview to a human-centered one.... The wellsprings of humanism were fed by many sources, but the meticulous (and, sometimes, not-so-meticulous) transcription, translation, editing, and annotation of texts were their legacy. The printing press enabled the standardization and dissemination of humanistic cultural corpora while promoting the further development and refinement of editorial techniques. Along with many other scholars, we suggest that the migration of cultural materials into digital media is a process analogous to the flowering of Renaissance and post-Renaissance print culture.[62]

What is left unstated in their paragraph is that the "post-Renaissance" lasted a long time. At Yale College, for instance, nontheological topics of study, especially science, did not become important until the nineteenth century, and even then they had to be carefully

The connection between communication theory and translation's radical transformation under the conditions of computation was already apparent in Warren Weaver's 1949 report titled "Translation."[71] Weaver's purpose was to explore the idea that one might design a computer program to translate texts from one language into another. Those familiar with Shannon and Weaver's text on the theory of communication will not find the following too surprising, but any bilingual person is likely to find Weaver's understanding of translation fantastical. Weaver wrote: "When I look at an article in Russian, I say, 'This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode.'" Weaver wrote this shortly after World War II, when protocomputers were first applied—with great success—to the problem of breaking Germany's military communication codes. In short, for Weaver, it was clear that computers were good for the tasks of decryption, so if a problem could be reconceptualized to look like a decryption problem, then it was probably something a computer could do. Despite skepticism voiced by scientific luminaries of the day,[72] Weaver's "Translation" essay was enormously influential[73] and, arguably, still informs computer scientists' approaches to translation. For example, the statistical approach to decoding that Weaver outlined in his essay constitutes the core of popular work in contemporary machine translation.

Two points of Weaver's text from 1949 are notable. First is the previously unusual idea that machines—specifically computers—can be used to translate texts from one language into another. Second is the radical translation of the very word "translation" that his text performs: Weaver equates translation with the operations of (information lossless) encryption and decryption. These two points will serve as points of contention for our exploration of the software arts.

Any bilingual person with access to the web can go online and test the current viability of Weaver's assertion that translation is just a form of decryption. For instance, find Google's online translation service[74] and try translating a text from one language into another. Unless the text is incredibly simple, the automatic machine translation will be riddled with errors, even gross mistakes. Certainly machine translation has improved dramatically since its inception in 1949, but there is still a gap between machine translation and good translation. Furthermore, there is not just a gap but a chasm between machine translation and transformation of a text in one language into a text in another language without loss or gain of information (i.e., a perfect translation as conceptualized by Weaver and other information theorists).

One of the problematics central to the software arts is to grapple with how computers and networks have come to incorporate this untenable idea of lossless translation and to pursue this by exercising older understandings of translation as they have been developed by scholars of the liberal arts and the humanities.

Translation and Science and Technology Studies

Historian and philosopher of science Michel Serres, in his book *Hermès III: La Traduction*, developed an approach to studying science and technology by means of translation.[75] Serres's method was then extended under the rubric of a "sociology of translation," alternatively, and more commonly, called "actor-network theory" or just ANT.[76]

Philosopher and science studies scholar Bruno Latour regrets the more recent and popular name "actor-network theory":

> At the time [the late 1970s and early 1980s], the word network, like Deleuze's and Guattari's term rhizome, clearly meant a series of transformations—translations, transductions—which could not be captured by any of the traditional terms of social theory. With the new popularization of the word network, it now means transport without deformation, an instantaneous, unmediated access to every piece of information. That is exactly the opposite of what we meant. What I would like to call "double click information" has killed the last bit of the critical cutting edge of the notion of network. I don't think we should use it anymore at least not to mean the type of transformations and translations that we want now to explore.[77]

Readers of Latour's more recent work know that his previous mention of "double click information" was far from casual.[78] In a recent book, he identifies "double click" as one of the fifteen "modes of existence" examined and strongly distinguishes it from other modes of existence, including what he calls the "mode of networks."

A sociology of translation is appropriate to an analysis of software, since computer scientists describe much of their work as translation. Beyond the problematics of machine translation (from one natural language into another), many practical problems are called "translations" in software design and engineering, including those translations alternatively labeled, in computer science, "compilations" or "interpretations" (as in the translation from a high-level programming language into a lower-level language) and "implementations" (as in the translation of an abstract system specification into a piece of working software).

Translation—as we know it in the arts and humanities—is always a force of change. Each translation involves either the loss or the addition of information, or both. Consequently, the result of any translation is not the same as the text translated—it is different. But translation—as it is known in computer science and information theory—is ideally lossless. From this perspective, a perfect translation from the source text to the target is one in which information is neither lost nor gained. A humanities scholar is likely to find the computer science approach to translation naive or idealistic. The computer scientist, conversely, might view the humanities approach as self-defeating since, if a translation can only be a deformation, degradation, or elaboration of the source

text, then an accurate translation is always out of reach. But humanists do not believe translation is impossible, only that any given translation can always be improved. This book explores the clashes that emerge from these differing perspectives on translation.

The Software Arts and the Liberal Arts of Language

Historian of science and technology Paul Edwards, commenting on a famous essay about computer historiography by Michael Mahoney (cited earlier),[79] explained that computer histories generally fall into a very small set of story types. Edwards wrote, "The first genre is an intellectual history in which computers function primarily as the embodiment of ideas about information, symbols, and logic." He continues: "The standard lineage here runs from Plato's investigations of the foundations of knowledge and belief, through Leibniz's rationalism, to Lady Lovelace's notes on Charles Babbage's Analytical Engine and Boole's *Laws of Thought* in the nineteenth century."[80] The lineage runs through the twentieth century and includes Alan Turing's machines, Norbert Wiener's cybernetic theory, the McCulloch-Pitts theory of neurons, and John von Neumann's collaborative work on computers. It is a history I retell in this book, but I retell it here in tension with two other less popular narratives: computer-as-rhetoric and computer-as-grammar.

This book is structured around the trivium, the three language arts of logic, rhetoric, and grammar. As Edwards and Mahoney point out, many computer histories tell the computer-as-logic story. In contrast, the computer-as-rhetoric history is one that emphasizes the role of computer software as a means of persuasion. Computer graphics and computer simulations are two forms of software widely deployed as forms of rhetoric. In contrast with the computer-as-logic and computer-as-rhetoric histories is the narrative of computer-as-grammar. Grammar concerns the rules of language. Grammar rules have long been considered by some to be machines. The story of computer-as-grammar came into focus in the mid-twentieth century, when it was ventured that the rules of language are machines or devices of software. By spinning together these three separate histories of computation, a clear picture can be woven of computing as a coproduction of the liberal arts of language.

The computer-as-logic, computer-as-rhetoric, and computer-as-grammar stories are three different interpretations of computing. These stories intersect somewhat but are also quite different. Their differences are partly explicable by the historical differences between logic, rhetoric, and grammar as three separate fields of study.

Our contemporary usage of the terms "grammar," "logic," and "rhetoric" is symptomatic of an old rivalry between them. As McLuhan describes in his history of the

trivium from ancient Greece to about the time of William Shakespeare, each of these areas of knowledge has been in competition with the others for millennia. When we praise someone for their logic and deride another for their rhetoric, we are voicing the current state of this competition. Clearly, today, logicians are accorded more respect than rhetoricians, since phrases like "empty logic" and "sound rhetoric" seem almost oxymoronic; the adjectives "logical" and "rhetorical" are laudatory and derogatory, respectively. Grammarians are now the leaders in this three-way rivalry, since grammar is institutionalized to the extent that we think nothing of sending our children to grammar school or having them learn grammar from their earliest years in school. We would be inhabiting a very different world if children were sent to rhetoric or logic school—rather than to grammar school—at age seven. Since rhetoric, logic, and grammar have been in a rivalry for ascendancy for centuries, the triptych painted in this book—of computer-as-logic, computer-as-rhetoric, computer-as-grammar—will expose some raw edges between them.

Stakes and Claims

The stakes of this book are threefold: pedagogical, industrial, and epistemological. First, if software is an art, then education needs to change to integrate it into the liberal arts. What do we teach? What do we learn? These are old questions that need to be posed once again in a world where basic literacy is not just a matter of English, Latin, and Greek but also of software. Second, if software can be written in the manner of an artist/humanist, then new avenues of software production beyond engineering and mathematics may be possible—avenues that some, like Steve Jobs, have already traveled. Third, if software is the new lingua franca, then there are a series of ethical and moral questions that must be pursued in conjunction with this epistemological transformation. What counts as knowledge, for whom, and at what cost?[81]

The most general claim of this book is that the software arts is a new name for something that has been going on for centuries: the pursuit of methods to invent and interrogate statements of connection, equivalence, and identity. Today, writing software is essential to both science and engineering. Yes, writing programs is very different from writing prose. Yes, computer languages are distinctively different from natural languages. But, regardless of whether we call software "machines" or "instructions," "objects" or "rules," regardless of whether we call the production of software "building" or "writing," "construction" or "composition," our names for software and its production are metaphors, and once we are working with metaphors, we are working as artists. I argue, along with Steve Jobs, that the arts are at the center of software.

History, Philosophy, and the Software Arts

I have tried to substantiate the constitution of the software arts both historically and philosophically. In what follows, I trace a more detailed history that runs from Denis Diderot, to Adam Smith, to Gaspard de Prony, to Charles Babbage, to Ada Lovelace, to Alan Turing, to today. When Denis Diderot and his collaborators on the eighteenth-century *Encyclopédie* were faced with the task of describing, in some standard manner, all of the processes, operations, and gestures employed in the studios and workshops of diverse artists and artisans throughout France, they needed to forge a language acceptable to the liberal arts and adequately descriptive of the mechanical arts. The encyclopedists did this by first listening to the artists and artisans to learn the language they used to describe their work practices and the machines they employed in their productions. In chapter 3, I call language like this "work and machine language." Then, the encyclopedists had to translate these various work and machine languages into a uniform lexicon and syntax for the *Encyclopédie* entries and a uniform visual language for the images that illustrated the entries. The language of the *Encyclopédie* had to cover everything from the making of stockings to the manufacture of pins. I argue that the encyclopedists' translation between the mechanical arts and the liberal arts eventually constituted the basis for what we know today as programming languages.

Philosophically, the substantiation of the software arts is entwined with this history. Francis Bacon, Gottfried Wilhelm Leibniz, and others were, in the words of semiotician, novelist, and historian Umberto Eco, engaged in a "search for the perfect language": a philosophical language, a "universal characteristic" with which all types of knowledge could be articulated. But none of them attempted a full-scale encyclopedia of knowledge. However, Diderot et al. did attempt such a full-scale catalog and did so with an understanding of how their project responded to Bacon's and Leibniz's aspirations.

The software arts continue with the encyclopedists' efforts to translate the mechanical arts into a language of the liberal arts and vice versa. The lingua francas forged for this project are now programming languages, forms of inscription liminally positioned between prose and machines.

On the Limits of Translation

Programming languages are limited in comparison with natural languages because they, and the programs they comprise, are imperative, independent, impersonal, infinitesimal, inscrutable, and instantaneous in ways that no previous forms of language are or have been. For example, in a programming language, one can "conjugate" only in

So, before one asserts that the computer is a brain, one might want to investigate, as Warren McCulloch and Walter Pitts did in "A Logical Calculus of Ideas Immanent in Nervous Activity," published in 1943, whether neural activities can be modeled as a set of logic circuits. If one wants to call a small computer a "smartphone," there is a large set of smaller equivalences that need to be established, for example to render and operationalize "buttons" for dialing a number as a graphical object on a touch screen. Equivalences between two unlike entities are established by translating one into the other.

These translations are frequently predicated on an older set of equivalences established through earlier translation efforts. Thus, for instance, McCulloch and Pitts's logical calculus was based on earlier efforts to translate all of mathematics into logic (e.g., the work of Bertrand Russell and Alfred North Whitehead). These earlier efforts, in turn, developed partially from translations attempted in the converse direction, such as George Boole's nineteenth-century efforts to render Aristotelian logic as a form of algebra or arithmetic.

There are many techniques and methods, especially from mathematical logic and the theory of computation, to equate objects and processes with software and to "prove" or "disprove" an equivalence between one piece of software and another and/or between a piece of digital software and an analog entity. Yet, from the perspective of the arts and the humanities, "proofs" of equivalence are not irrefutable but rather are only arguments or demonstrations that a translation was done rigorously. In chapter 6, on rhetoric, various forms of demonstration with computation are examined.

If we think of these equivalences as "proved" and thus settled once and for all, we are unlikely to think of alternatives or innovate further. If instead we think of these equivalences as the result of a translation effort, we can always ask, "What got lost in translation?" For example, even though Claude Shannon "proved" in his master's thesis that Boolean logic and Boolean circuits are the same thing, if we persist in asking how they are different, we quickly unearth a set of problems circuits have that written logics do not; for example, while designing circuits, we have to worry about heat produced by the electrical current running through the circuit. While modern computers need fans to keep them cool, George Boole's nineteenth-century arithmetic of logic never needed a fan! Differences in materiality are significant differences. This example is discussed in detail in chapter 5, on logic.

Translation as Imperfect

To take this liberal arts attitude about the messiness of translation into the technical literature entails closely rereading pivotal papers of science, mathematics, and engineering to find and evaluate what got lost in translation. Given a proposed equivalence

X = Y, what link was established between X and Y; how was X translated into Y and Y into X? What had to be put to the side or ignored in order to establish the equivalence? Chapter 2, on translation, is essentially a chapter on methodology—an interpretation of actor-network theory—and its employment in following such chains of equivalence in the texts of software.

Bruno Latour wrote a small philosophical work, *Irreductions*, that starts with a declaration as far afield from a digital ideology as one can find. Latour asks, "What happens when nothing is reduced to anything else? ... Nothing is, by itself, either reducible or irreducible to anything else. I will call this the 'principle of irreducibility,' but it is a prince that does not govern since that would be a self-contradiction."[82] While a digital ideology might include a myriad of reductions showing how life, the universe, and everything can be digitized or reduced to some form of computer program, Latour's declaration establishes an alternative and opposing manifesto: everything and everyone is singular, nothing is equivalent to anything else, everything is irreducible. However, even if that is the case, things can still be connected or linked together. With enough links between things, we have a network, thus "actor-network theory."

Under the theory of ideology exercised in this book, inspired by Algirdas Julien Greimas, an ideology can be analyzed as a set of equivalences between a number of ideas and/or identities. These equivalences and/or inequalities can be stated in a number of ways, including copular statements (X is Y), equations (X = Y), and—especially important for the digital—assignment statements (X := Y; i.e., X is assigned the value of Y) and rewrite rules (X → Y; i.e., X is rewritten as Y). To do actor-network theory on an ideology is to question its equivalences: to ask whether they are really definitional or are provisional. In a sense, it is akin to what Socrates did in the agora of Athens by flipping the copular assertions of common sense and turning them into questions of definition: What is courage? What is truth?

Actor-Network Theory and Software Studies

Two independent inspirations for ANT were Greimasian semiotics[83] and sociologist Harold Garfinkel's ethnomethodology.[84] The work of ethnomethodologists has been dominated by ethnographic concerns and methods—observing and writing down the oral and physical interactions between people. In contrast, Greimasian semiotics was originally developed to study texts of literature.

ANT has been applied to the equations and equivalences of many areas of science and technology but only rarely to the texts of software. When they have been interested in software, the way that ANT practitioners, in particular, and STS researchers,

more generally, have pursued the study of computer science and software engineering has been heavy on ethnography and light on semiotics. Consequently, there are a number of wonderful ethnographic studies of various software development projects in which we learn a lot about the programmers and how they interact, but these studies tend to leave out of focus the texts of software: What exactly was the code being written by the programmers? What were the technical papers being read and written by the project group members, and how did the ideas proposed in the papers make their way—or not—into the goals and accomplishments of the programmers' productions, the code and its commentary?

This book illustrates how ANT, the sociology of translation, can be used as a method for looking at the texts of software (including code and technical papers). It is a necessary complement to the STS work that has been done to understand the people producing the software; that is, to, as one of Bruno Latour's books is subtitled, "follow scientists and engineers through society."[85] We need to study the texts of software as well as follow the people who produce it.

One would think that computer historians would have already written a lot about software, but much of computer history has focused on hardware, and only recently has software become an object of study. One might equally imagine that philosophers and historians of logic would have written a lot about software, but they rarely investigate contemporary software texts. Thus, in the lacunae of STS, computer history, and the history and philosophy of logic, there is an opening for a new kind of scholarship that focuses on the texts of software. This new kind of scholarship is the emerging field of software studies, and this book can be seen as a contribution to it.

Close Readings

Simply put, this book is a close reading of key texts of computer science and its history. For example, in chapter 4, when we read closely Donald Knuth's features of an algorithm—perhaps the key term for computer science—we find a circularity and several logical inconsistencies; for example, Knuth claims that algorithms can be defined independently of any programming language, yet all of his algorithms are defined in terms of a specific programming language of his own design. In chapter 2, reading Alan Turing's paper on the definition of Turing machines and papers by his students and colleagues popularizing the idea, we find that central to Turing's original paper is a negative result—that there are tasks that the computer cannot be programmed to perform. Yet, in its popular reception, we find an enthusiasm for the idea that a computer can be programmed to do anything and everything. In chapter 5, by closely reading texts

on logic and computation, we find—directly at the roots of software—a long-standing and strangely circular project to make logic into arithmetic and arithmetic into logic. This circular project has profoundly reshaped logic and has shaped computing since its inception. In chapter 6, on rhetoric, we follow the radical transformation of what it used to mean to demonstrate a point in a rigorous argument into the very different "demos" of today, the demos we see in Silicon Valley and in the arguments of "big data" practitioners. I call these demos "abductive demonstrations" and contrast them with the previously dominant forms of deductive and inductive demonstrations. In chapter 7, in an examination of the work of Noam Chomsky—as well as his teachers, colleagues, and students—we encounter the nascence of the notion that theories can be "devices," specifically devices rendered in software. When theories become devices, we do not ask *why* questions, we ask *how* questions: we ask whether the theories work. But we should also ask for whom they work and at what cost.

In sum, when we read the technical texts of software closely, we frequently find their popularizations are miles away from what the original texts actually say. Culturally, economically, politically, socially, and technically, this would not matter if software were a marginal concern left to a few specialists with no power and no influence, but today software looms large in research and teaching and in the everyday lives of people all over the world.

The Organization of the Book

The proposal of this book is that we read and write software as an extension of the liberal arts, specifically the trivium, the three language arts of the liberal arts. Logic, grammar, and rhetoric (in the guise of the "demo") are all instantly recognizable as intrinsic to software and central to computer science. The book is organized around these three liberal arts but with the first four chapters—this introductory chapter and then chapters on translation, language, and algorithms—devoted to introductory materials. In chapter 5, the first of the trivium, logic, is examined. Following that is a chapter on rhetoric, and then a chapter on grammar.

The composition of the book is self-similar, with the same kind of arguments made for the book as a whole and also at the scale of the chapter. I closely read a piece of computer science, looking for its instabilities and contradictions and seeking clues to its historical precedents. I then chase down those historical precedents to see what lost or neglected alternatives existed that could serve as improvements or correctives to the computer science of today. The instabilities and contradictions usually result from efforts to reduce everything to mathematics or logic or, more specifically, to digitize,

to turn everything into a form of arithmetic, to collapse it into a concern of calculation. In contrast, the historical alternatives I have found come from the arts and the humanities.

I have been writing this book with two readerships in mind. One is a set of cultural workers, artists, and scholars of culture interested in examining what software might have to offer in terms of theories, methods, or tools. The other is a group of computer scientists and software engineers whose work is bound up with cultural production—game design, social media, or streaming video. My message to all readers is that culture and computing are knotted together, and one way we can understand their entanglements is by closely reading the texts of software—code and technical books and papers. For each chapter of the book, I have a hope for the reader.

To begin, if this chapter, the introduction, has worked as I wished, the reader is willing to entertain the possibility that although computing can be seen as science (e.g., computer science) and as engineering (e.g., software engineering), it can also be seen as an art, or a collection of arts: the software arts.

Chapter 2, on translation, is offered as a "methods" chapter. Translation is known to the scholar and to the computer scientist, but each is familiar with a very different flavor of it. The main example discussed in the chapter is a set of texts from the beginnings of the theory of computation, concerning Alan Turing's machines, Alonzo Church's lambda-calculus, and popularizations of the Church-Turing thesis that claim that there are no limits to what a computer can do. This popularization is not true. Turing's original publication shows definitively that computers do have limits. By reading popularizations of the texts of software as a series of translations—from the most technical to the most popular—I show how the popular reception of a technical text can result in a fantasy that contradicts the findings of the original publication. My hope is that the reader will see how to reframe a popularization as a series of translations from the technical literature "out" into the wilds of popular culture and then back again—into the technical literature. The methodology presented is an amendment and an extension to actor-network theory, also well known in the field of science and technology studies as the sociology of translation. This contribution to actor-network theory (or ANT for short) is the main methodological contribution of the book.

In chapter 3, on language, I argue that computers are not information technologies and that the operations of computing are not the functions of mathematics. To expand on these assertions, I narrate a history of programming languages that starts in the artisans' workshops of eighteenth-century France. I trace a history of the division of labor as it was practiced in these studios and workshops; as it was transcribed by economist Adam Smith in the first chapter of his book *The Wealth of Nations;* as Gaspard Prony

abduction is a form of guessing. Alternatively, we might say that abduction is a form of interpretation, a practice well known to the arts and the humanities. The chapter proceeds from older means of making a point to the newest forms of persuasion. I hope to provide the reader with ways to both question and compose software-based arguments.

Chapter 7 is on the third of the three liberal arts of language: grammar. For a long time, grammar was a political project prosecuted as pedagogy in order to homogenize written and spoken language of empires. Later, it was deployed in an analogous manner to consolidate nation-states. Grammar was initially predominantly prescriptive. Then, in the late nineteenth and early twentieth centuries, grammar was reframed by linguists desiring to describe how language is actually used. With linguist and semiotician Ferdinand de Saussure, grammar became descriptive. When it did, its locus moved from textbooks into machines—both mechanical and imagined mechanisms of the brain. By the mid-twentieth century, linguistics had joined forces with the mathematical formalism championed by David Hilbert. This resulted in a transformation of linguistics to exclude meaning from its object of study. In the words of Noam Chomsky, "The study of meaning and reference and of the use of language should be excluded from the field of linguistics."[89] Instead, Chomsky and his followers pursue linguistics in the form of meaningless syntactic manipulations ultimately articulated as computer programs. After Chomsky, grammar machines became software, and claims were made that software could constitute a theory of language. This represented a huge shift in intellectual culture. When a computer program, a piece of software, can be a theory, we have entered what I will call the "computational episteme." In a computational episteme, software is taken for theoretical insight, and meaning is pushed to the margins. These conditions are strange and challenging. I hope that the reader will see that one way to make sense in a computational episteme—to revive meaning—is to act as an artist, to engage in the software arts.

2 Translation

The first part of this chapter is an open-ended discussion of translation and its uses in several areas of study, especially its relevance and definition in the arts, the humanities, science and technology studies (STS), and computer science. In STS, the analysis of translation is frequently pursued under the rubric of "actor-network theory" (ANT) and employed to study a diverse range of scientific fields and technical objects. We will see how the ANT methodology can be amended and extended for the study of software.

The second part of the chapter focuses on a detailed example pivotal for the foundations of computer science and now central to popular opinion: David Hilbert's "decision problem" and its analysis by Alan Turing and Alonzo Church. Through a close reading of some key texts that address Hilbert's decision problem, I will illustrate how to use the methods of ANT to scrutinize the formal mathematical/logical proofs of this technical literature. I focus on the specific means—the techniques of translation—deployed by Turing, Church, and others to assign identities and equivalences, especially those that assert that computers are like, or the same thing as, people. My point is that there are always gaps that separate the computer from whatever or whoever it is rhetorically equated with. I implore you—as do signs on the platforms of London Tube stations—to mind the gap!

After examining how equivalences are forged in the more technical language of the foundations of computer science, I will pursue Hilbert's decision problem as it emerges from the technical literature and gets translated into larger, more popular venues where false assertions are made, such as "computers can be programmed to simulate anything." A reworking of the ANT methodology provides the tools we need to understand how the popularization of the texts of software establishes stronger and weaker links between the technical works and everyday common sense about the limits of computing.

Assignments and Simulations

In Pedro Almodóvar's 2011 film *The Skin I Live In*,[1] Robert Ledgard, a plastic surgeon, tracks down a young man, Vincente Piñeiro, holds him hostage, and—over the course of six years—subjects him to a series of operations that transform him into a simulation of Robert's late wife, Vera Cruz. When one of Robert's colleagues accuses him of performing Vincente's sex-reassignment surgery without Vincente's consent, Vincente attests that s/he has been a willing participant, but when Robert tries to have what initially seems to be consensual sex with Vera/Vincente and s/he takes Robert's gun, kills him, and escapes, we realize that, like Robert, we have been outsmarted by Vincente's skills as a simulator. Vincente never identified with Vera and never bonded with his captor, Robert. Rather, Vincente just plotted strategically and waited for the right moment to strike back. At the end of the film, Vincente returns to his mother to reclaim his identity as her son.

Robert's surgeries make Vincente into a simulation. After six years of cutting and grafting, s/he looks and sounds like Robert's late wife—but is not. Almodóvar's movie tells us that the essentials of identity are more than skin deep. In the fiction, Vincente is certainly not Vera; he is a simulator who only pretends to accept the role of Vera. The crucial plot point pivots on the critical difference that, inside, Vincente is not and never can be Vera. Robert cannot cut and stitch together his late wife from the flesh of Vincente, because he can never take the memories and suffering of the man out of the woman.

A simulation can never be its model. Most contemporary simulations are digital productions—not heinous crimes of the flesh. Nevertheless, just as Vincente is crucially different from Vera, any digital simulation always incorporates a set of crucial differences that separates it from its model.

In fiction, Almodóvar's monstrous simulation is more the rule than the exception. Since at least Mary Shelley's *Frankenstein*,[2] the simulation that falls short of its model has been a frequently told story. These stories scare us by making the familiar unfamiliar: Vincente looks like Vera but is not. Dr. Frankenstein's creature looks like a man but is a monster. In philosophical aesthetics, there is a word for this kind of scariness. Sigmund Freud called it the "uncanny."[3]

Fetishism and Disavowal

But what happens if we recognize the crucial difference between the simulation and the model yet act as if no substantial difference exists? Freud has another term for this condition: "fetishism."[4] The fetishist is able to know the difference between simulation

and model and, simultaneously, believes them to be the same. The fetishist says, I know very well the differences, yet I will overlook them. It is a condition that, according to Freud, is closely linked to disavowal, in which we refuse to recognize a traumatic event. Robert is a fetishist. He knows very well that Vincente is not Vera, but he disavows his traumatic criminal and surgical interventions in order to live in a fantasy where he imagines his former wife, Vera, to be reincarnated.

Those who create or perform simulations were accused of immoral behavior long before Freud's time. In ancient Greece, Plato accused the rival Sophists of being practitioners of simulation because of the way they practiced rhetoric as a form of false representation. "Simulation" was a dirty word for thousands of years before it became what it is now: a respectable pursuit in science, engineering, and mathematics.

Certainly in Robert's case, the simulation of Vera through Vincente is immoral and criminal, although not all simulations are immoral constructs and not all fetishists are criminals. This state of disavowal is the norm of contemporary life, and most of us are indeed nonviolent fetishists. We know very well that the digital simulations of our everyday life are not the same as the models they are based on. Our email, our smartphones, our streaming movies and videos, our social-networking "friends," and even the synthesized voices and the digitally filtered vocalists we hear on the "radio" are all digital now but based on precomputer models, media, and institutions of the past. We treat these simulations as though they were essentially equivalent to their respective models. This is everyday digital life. It is reproduced with hype, disavowal, and ignorance.

I use the term "model" in a manner analogous to the way biomedical researchers use the term. For example, a cancer researcher can use mice as a model organism to test out a new drug because mice share enough similarities with humans to illustrate how the drug might perform on humans. Models, in this sense, are not closer to some ideal than simulations are. Thus, when I assert that a simulation can never be its model, I am not saying that simulations are "virtual" and models are "real." Nor am I expressing nostalgic sentiments imagining that models are "authentic" and simulations are "fake." Mice are neither more real nor less authentic than anything else. Rather, for the purposes of biomedical research, mice make viable models because they resemble humans in some important ways and, obviously, are quite different in other ways.

I posit that there is no ordering on models and simulations: neither is closer to the real than the other. In contrast, as we will see in chapter 6, on rhetoric, others, like Plato, posit that simulations are much further away from the real.

Identity, Equality, and Assignment

On her 1982 hit album *Big Science*,[5] Laurie Anderson sang, "I met this guy—and he looked like he might have been a hat check clerk at an ice rink. Which, in fact, he turned out to be. And I said: Oh boy. Right again. Let X=X." Anderson's quirky humor here ties her to a long tradition in philosophy. Let X=X is a statement of the law of identity, the first of the three classical laws of thought, originally written by Plato and subsequently elaborated by many philosophers.

As any computer programmer knows, assignment is not a declaration of identity. Assignment is a form of coercion that is time dependent. In mathematics, X=Y is a declaration of identity that means X is Y and always will be, and equality is commutative: writing X=Y means the same thing as writing Y=X. Unlike in mathematics, in computer programming, X:= Y is an assignment that means X becomes Y after a certain moment in time. Assignment is normally not commutative: writing X:= Y usually means something entirely different than writing Y:= X.[6]

As a statement, X:= Y hardly seems as if it could function as the kernel of a compelling story, but of course the dramatic potential of identity is more in the suspense around its assignment than in its declaration. Assignment can be dramatic under conditions when it is not warranted, not wanted, or not expected. In Almodóvar's movie, when Vincente is surgically reassigned to be Vera, it is done under conditions of extreme physical and psychological violence. To reassign Vincente's identity, Robert must exercise tyrannical and terrible power.

A major concern of this book is assignment and its entanglements with identity and equivalence. Specifically, the focus here is digital assignment. How and by what means are computers used to assert that different things—the model and the simulation—are the same?

The computer's role in contemporary questions of assignment and identity is a starring one, but there are many other players, too, especially us. In acknowledging our role, I am not making claims about "us" like those made by writers who want to argue that the younger generation are "digital natives"[7] or that computers are making us stupid.[8] Instead, at issue are the specific computational means deployed to assign identities and equivalences, a means that I refer to more generally as simply "translation."

I do not yearn for a predigital, authentic time. Rather, my motivation comes from a belief in education even though the hype and disavowal of our contemporary digital conditions cannot be addressed with education. Even well-educated people fall for hype and indulge in disavowal. However, I do think the third factor of digital fetishism—ignorance—can be treated educationally. Thus, my hope is that a careful analysis of

this project was philosopher and mathematician Gottfried Leibniz, who was also the co-inventor (with Isaac Newton) of the calculus.

Many highly readable books of computer history draw a direct lineage from Leibniz to today by passing through some of the major figures of mathematical logic. For example, Martin Davis's excellent book *The Universal Computer: The Road from Leibniz to Turing*[16] jumps from Gottfried Leibniz (seventeenth century) to George Boole (nineteenth century) and then on to Gottlob Frege, Georg Cantor, David Hilbert, Kurt Gödel, and, finally, Alan Turing (twentieth century).[17] When the inventor of cybernetics, Norbert Wiener, named Leibniz the "patron saint" of cybernetics,[18] he was telegraphing this now frequently told history of the efforts made, from Leibniz to Turing, to craft a Baconian, artificial language tantamount to a machine. How can language work like a machine? How can a machine do the work of language? Any answers to these questions depend on an understanding of how Leibniz and his successors approached the problems of translation—for example, the problem of translating logic into a machine.

Today, computer science has answers to these questions. The Leibnizian/Baconian artificial language has become a programming language. The linguistic expression that is tantamount to a machine is software, and the means of translating from language to machine is alternatively called, in contemporary computer science, "compilation" or "interpretation" of a programming language.

The roots of the computer science ideal of (information) lossless translation can be described as Leibnizian, and, from the perspective of this tradition, Warren Weaver's fantasy of translation as decryption (mentioned in the introduction) seems no more fantastical than the idea that computer programs written in a programming language can be translated into a machine or, more specifically, into electrical currents in the circuits of a computer.

French logician, mathematician, and linguist Louis Couturat noted that Leibniz "thus conceived logic, in turn, in the form of an arithmetic, an algebra, a geometry, even a mechanism. Each, moreover, entirely symbolic and constitutive of concrete expressions of the same abstract science. The imaginative idea of so transposing logic and casting it into mathematical forms stemmed from, on the one hand, his desire to render reason tangible and palpable and from, on the other hand, his deeply held conviction in the harmony between all rational sciences that, according to his favorite expression, must 'symbolize' between them."[19] The translations Leibniz was able to effect—for example, from the operations of arithmetic to the mechanical operations of a machine—seem sensible and circumscribed. Leibniz's successors have managed much grander translations, but this intellectual tradition demands very careful proof for each translation claimed, so even these grand successes are carefully circumscribed.

Digital Ideology and Digital Life

More precisely, such translations are *usually* carefully circumscribed. A counterexample can be seen in the claim by the Turing Award winners and cofounders of the field of artificial intelligence, Allen Newell and Herbert Simon, that humans and computers are equivalent because both are "symbol systems." They attempted to demonstrate this equivalence by implementation: they tried to write software that could think like humans.

Newell and Simon are far from lonely in a cohort of computer scientists who have made hyperbolic claims about what computers are or what computers can or cannot do. Notoriously, many artificial intelligence researchers have been promising for decades that human-level, even superhuman-level, intelligence is just around the corner—and they continue to do so today. There is a huge body of literature that is, essentially, science fiction published not as entertainment but as purported scientific fact. Included in this literature of science fiction are claims that people are computers, that our brains are computers, that the entire physical universe is a big computer, and on and on. Each of these "scientific" claims has fictional precedents in which writers have envisioned, for example, sentient, conscious robots.

Bordering this territory of science fiction is a set of statements framed as scientific hypotheses about the translation of various phenomena into computational terms. For instance, Howard Gardner, in his overview and introduction to cognitive science, states that one of the paramount features of cognitive science is this belief: "There is the faith that central to any understanding of the human mind is the electronic computer. Not only are computers indispensable for carrying out studies of various sorts, but, more crucially, the computer also serves as the most viable model of how the human mind functions."[20]

So, for some cognitive scientists, the human mind is a computer; for many molecular biologists, the genetic code is a computer code; and so forth. These are not fanciful leaps of faith but rather beliefs held by large groups of scientists who publish in peer-reviewed journals. Let us lump this together with the science fiction and label the lot "digital ideology" to distinguish these beliefs in computation from something far more pervasive: "digital life."

Digital life moves outside of professional circles and beyond the technical vocabularies of specialists' dialogues. Digital life is everyday life in a society that enacts a digital ideology by replacing everyday institutions (e.g., the mail system, the cinema, banking, etc.) with technologies that incorporate computers in a manner that makes them indispensable. For example, what would be left of your personal photo collection if you lost the digitalized, electronic archive? Do you have paper backups?

Digital life is lived with a belief in the equivalences of digital ideology. Digital ideology is founded on very many assertions of equivalence; among these assertions are that the brain is a computer or that digital video is just like older forms of film. In the scientific literature, digital ideology is usually elaborated in very careful, tightly circumscribed, and rigorously argued equivalences ultimately presented as identities: X=Y. These equivalences take the form of technical claims. In the technical literature, one cannot easily claim, for instance, that logic is a form of arithmetic. Rather, such a claim has to be substantiated using a set of techniques of scientific and/or mathematical demonstration; that is, techniques of translation to prove the equivalence of two entities.

Yet even if the equivalences, the identities, established through the translations of science and technology are correct, they are not perfect. We will show how the technical equivalences established by computer science are imperfect, and we will describe what is lost or added when these equivalences are established as identities.

The Computational Condition = Digital Ideology + Digital Life

In his 1979 diagnosis of the current state of knowledge for the government of Quebec, French philosopher Jean-François Lyotard predicted an immediate future in which "the direction of new research will be dictated by the possibility of its eventual results being translated into computer language. The 'producers' and users of knowledge must now, and will have to, possess the means of translating into these languages whatever they want to learn."[21] Lyotard labeled his diagnosis—that we have turned away from other languages (especially narrative language) to computer languages—the "postmodern condition." Lyotard's postmodern condition might be more precisely called a computational condition, a condition that encompasses both digital ideology and digital life. In my phrase "computational condition," I expect humanities scholars will hear the echo of both Jean-François Lyotard's "postmodern condition" and Hannah Arendt's "human condition."[22] A computational condition is a state of knowledge in which scientifically and technically established equivalences are taken to be perfect identities without loss or gain on either side of the translation. They are equivalences taken for identities and therefore sound like absurd tautologies when they are said together: May I have a glass of H_20 water?

The equivalences of digital ideology are taken to be culturally conditioned common sense if, once demonstrated, they are repeated widely and often enough. According to this conception of ideology and common sense, common sense is dynamic and subject to change as new ideas and technologies become popular. In the words of philosopher

Antonio Gramsci, "Every social stratum has its own 'common sense' and its own 'good sense,' which are basically the most widespread conception of life and of men. Every philosophical current leaves behind a sedimentation of 'common sense': this is the document of its historical effectiveness. Common sense is not something rigid and immobile, but is continually transforming itself, enriching itself with scientific ideas and with philosophical opinions which have entered ordinary life.... Common sense creates the folklore of the future, that is as a relatively rigid phase of popular knowledge at a given place and time."[23]

Digital ideology is employed in everyday digital life when narrow, technical, circumscribed equivalences are loosened and expanded. For example, those of us who live in a computational condition see no contradiction in treating an email to our family as the same thing as a written letter sent through the postal service and delivered by hand. When we watch a movie on a streaming video service, we say we have seen the movie, even if the small-screen experience and the former conditions of cinema-going (popcorn, crowds, paper tickets) are quite different. More subtly, it is now unclear what we have done—a digital act or a physical act—when we say we have been "searching" for someone or something or when we say we are "friends" with someone.

From Media Studies to Actor-Network Theory

So how do the equivalences of digital ideology come to be used, loosened, and expanded into digital life? Some forms of science and media studies and newer forms of social media analysis employ a model of "diffusion" of ideas or envision the movement of ideas with the metaphors of epidemiology or genetics.[24] Ideas are said to "spread like a virus"[25] (thus the phrase "viral media") or to move like genes through reproduction and evolution (thus the notion of "memes").[26]

Many older schools of communication and media studies also trope people as physical or mechanical systems. For example, the regnant metaphor of mass communications is the "mass" or physical system. This metaphor allows the researcher to ask questions like, "What is the *impact* of a given message on an audience?"[27]

In the 1940s, Robert Merton and Paul Lazarsfeld[28] advanced a program of research in which social structures were seen to be stable or unstable, in equilibrium or disequilibrium, according to group dynamics and the media messages that influence the members of a group. The metaphor of people as a thermodynamic system engenders questions about the production and breakdown of social order.

All the metaphors of old and new media studies intended to explain how ideas move are inaccurate and mildly insulting because, ironically, they implicitly assume

that people do not have language—that people do not speak, listen, read, or write. These notions of diffusion, contagion, spreading, impact, equilibrium, and evolution all figure people as mute, nonhuman animals, organic or inorganic materials. Obviously, people are not inert masses moved only by Newtonian mechanics. Nor are they exclusively porous materials through which ideas are "diffused," vectors for viral "contagions," or breeding animals restricted to reproducing their "genes" or "memes"! In contrast to these old and new schools of communication and media study, let us start with the much more reasonable assumption that people think and speak.

Developers and practitioners of actor-network theory—the sociology of translation—have shown quite definitively why these other approaches are inadequate to the task of explaining how ideas become a part of everyday life and common sense. Instead of this pack of metaphors, actor-network theory hypothesizes that the bridge from ideology to everyday life is a set of cognitively, culturally, and materially specific practices of translation that are employed rhetorically and technologically to convince people of equivalences, differences, and identities.

Interestingly, with few exceptions, actor-network theorists have not studied software.[29] When science studies researchers have focused on software, they have not examined the texts themselves—code and technical articles from the literature of computer science. Rather, they have followed the people who produce and use software.[30]

This lack of attention to the texts of software is surprising because actor-network theory is said to combine the concerns of ethnography and ethnomethodology with insights from semiotics, the humanistic study of meaning and meaning making, a set of methods developed especially for textual analysis. I will employ an understanding of translation that is deeply indebted to actor-network theory, but to supplement ANT for the purposes of studying the texts of software, I will need to draw from the even deeper well of translation as it has been practiced in the humanities.

Translation and the "Ductions" of Michel Serres

Philosopher and historian of science Michel Serres wrote, "We only understand something according to the transformations that can be performed on it. There are at least four such transformations: deduction in the area of logic and mathematics; induction in fields of empirical experimentation; production in domains of practice; translation [traduction] in the space of texts. It is not unexpected that they all incorporate the same root word. There is no philosophy except for that of "duction"—with a variety of prefixes. A lifetime might be spent trying to clarify this state of affairs."[31] This quotation is from the beginning of Serres's book on translation, *Hermès III: La Traduction*,

Latour's DNA and the Double Helix

Latour illustrates this more complex version with a comic strip in his book *Science in Action: How to Follow Scientists and Engineers through Society*.[34] The comic strip depicts the fate of a statement about deoxyribonucleic acid (DNA) in, at the beginning, reverse chronological order, and then, at the end, in chronological order. In sum, it shows how we can question scientific facts by looking at their earlier formulations and then shows how those earlier, conditional formulations are developed into definitive, textbook statements of equivalence.

In the first frame of the comic strip is written: "The DNA molecule has the shape of a double helix." Latour writes,

> To sketch the general shape of this book, it is best to picture the following comic strip: we start with a textbook sentence which is devoid of any trace of fabrication, construction or ownership; we then put it in quotation marks ["The DNA molecule has the shape of a double helix"], surround it with a bubble, place it in the mouth of someone who speaks; then we add to this speaking character another character to whom this character is speaking; then we place all of them in a specific situation, somewhere in time and place, surrounded by equipment, machines, colleagues; [First Colleague: "Why don't you guys do something serious?" Second Colleague: "Maybe it is a triple helix." Third Colleague: "It is not a helix at all." Watson to Crick: "If it had the shape of a double helix ..." Crick to Watson: "... this would explain Chargaff ..." Watson: "... and it would be pretty."] then when the controversy heats up a bit we look at where the disputing people go and what sort of new elements they fetch, recruit or seduce [note Latour's use of "se-duce," a term also preferred by Serres] in order to convince their colleagues; then we see how the people being convinced stop discussing one another; situations, localizations, even people start being slowly erased; [Colleague to several other colleagues: "They say Watson and Crick have shown that DNA is a double helix." In the penultimate frame of the comic strip, no people are shown, just a book with this sentence highlighted in it: "Watson and Crick have shown that DNA is a double helix."] in the last picture we see a new sentence, without any quotation marks, written in a textbook similar to the one we started with in the first picture. This is the general movement of what we will study over and over again in the course of this book, penetrating science from the outside, following controversies and accompanying scientists up to the end, being slowly led out of science in the making.[35]

With this comic strip and this paragraph, Latour explains how he plans to lead us, the readers of the book—first from the outside to the inside of science, and then, once inside, to lead us by following the scientists. He then tells us that we will be "slowly led out of science in the making." To move from within science in the making to the outside is to move, for instance, from scientific journals with a small readership to textbooks with a large readership. In a metaphorical/etymological sense, we can say that Latour's text and comic strip describes first an "introduction," a translation that leads