A Thesis
Submitted to the Faculty of the
Graduate School of Arts and Sciences
Of Georgetown University
In partial fulfillment of the requirements for the
Degree of
Master of Arts
in Communication, Culture, and Technology

By

Olivia F. Payne, B.A.

Washington, DC
April 25, 2016
From dawn to dusk, humans are involved in abstraction and expression. As symbolic thinkers, humans are in a constant state of meaning-making. In machinic computation we implement these meaning-making abstractions in algorithmic form in order to automate them. But even if our "hand-wrought" computation trails automated efforts by orders of magnitude, computation -- intrinsic to meaning making -- remains a quintessentially human project. Why, then, do we invariably see "computation" as necessarily dehumanizing? Why has computation become, for many, the exclusive domain of science, technology, mathematics and engineering? If meaning-making through abstraction, abduction, and expression is a common feature of human experience, why do we view computation as alien and extrinsic?

It is a socio-historical accident that computing has been narrowed recently to involve only certain disciplines. If we accept that the process of symbolic representation has been intrinsic to human thought since the emergence of Homo sapiens, and that machinic computation is roughly the further abstraction and automation of that process, we see that the common trope of dehumanization deserves to be turned on its head: In other words, it is not that computers threaten to drain us of our humanity, but that computers may be understood as extending and amplifying that most human part of ourselves.

In engaging in symbolic representation and abstraction, both with and without computational mediation, I design textiles and make quilts in order to investigate a new theoretical framework for computational thinking. Through displaying coded patterns in quilt form, I highlight the importance of thinking creatively about computing and pushing others in the humanities to think beyond the current conception of computation.
The research and writing of this thesis

is dedicated to 10 PRINT, my original inspiration.

10 PRINT CHR$(205.5+RND(1)); : GOTO 10, MIT Press, a book
about a one-line Commodore 64 BASIC program was
 collaboratively written by Nick Montfort, Patsy Baudoin,
John Bell, Ian Bogost, Jeremy Douglass, Mark C. Marino,
Michael Mateas, Casey Reas, Mark Sample, and Noah Vawter.

This book, and the study of 10 PRINT, introduced me to
creative computing and to a consideration of code beyond its
functional purpose. Source code can be understood as an
artifact consistent with thousands of years of human effort,
not a radical break from it.

-Olivia F. Payne
ACKNOWLEDGMENTS

I would like to thank numerous people who have assisted and provided guidance along the way. My sincerest of thanks to my thesis advisors, Garrison LeMasters and Martin Irvine, who provided expertise, probing questions, and helpful comments when I needed them most. Thank you to Dr. LeMasters for the stimulating course work and discussion that led me to and through this project, and to Dr. Irvine for pushing my work further, into a different, better place. A special shout out to Diana Owen, J.R. Osborn, Sarah Twose, Chris Miller, Linda Huber, Jaime Gonzalez-Capitel Martorel, Nimmy Muraleedharan, Nathan Danskey — “CCT-ers” have a unique place in this world and in my heart. And to my loved ones, Rebecca Stefanon, Dyon Stefanon, Kunj Patel, Alex Clinton, Caitlin Fitzpatrick, Zach Payne, Mariann Payne, John Payne and Molly Monahan — your encouragement and endless patience is a blessing. Thank you for your support, your insights, and your belief in the project. It means the world.
# TABLE OF CONTENTS

I. Introduction.................................................................................................................. 1  
II. Symbolic Representation............................................................................................ 13  
III. Computing and Computational Thinking............................................................... 19  
IV. Making as a Form of Inquiry...................................................................................... 26  
V. Code / Quilt.................................................................................................................. 35  
VI. Conclusions & Future Work...................................................................................... 44  
   Appendix......................................................................................................................... 47  
   Bibliography.................................................................................................................. 58
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Figure Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>quiltOne</td>
<td>13</td>
</tr>
<tr>
<td>4.1</td>
<td>simpleEllipse</td>
<td>30</td>
</tr>
<tr>
<td>4.2</td>
<td>simpleEllipseColor</td>
<td>31</td>
</tr>
<tr>
<td>4.3</td>
<td>simpleEllipseRepeat</td>
<td>32</td>
</tr>
<tr>
<td>5.1</td>
<td>Core Rope Memory</td>
<td>36</td>
</tr>
<tr>
<td>5.2</td>
<td>quiltTwo</td>
<td>42</td>
</tr>
<tr>
<td>A.1</td>
<td>basicLines</td>
<td>48</td>
</tr>
<tr>
<td>A.2.1</td>
<td>basicPlaid 1</td>
<td>50</td>
</tr>
<tr>
<td>A.2.2</td>
<td>basicPlaid 2</td>
<td>51</td>
</tr>
<tr>
<td>A.2.3</td>
<td>basicPlaid 3</td>
<td>52</td>
</tr>
<tr>
<td>A.3</td>
<td>brokenEggs</td>
<td>54</td>
</tr>
<tr>
<td>A.4</td>
<td>bezierFlowers</td>
<td>57</td>
</tr>
</tbody>
</table>
I. INTRODUCTION

From dawn to dusk, humans are involved in symbolic abstraction and expression. As symbolic thinkers, humans are in a constant state of meaning-making.\(^1\) This meaning-making through symbolic representation, patterns of symbols that stand for something, is referential to a symbolic past, present and future, and is generative and combinatorial over time. The word or sign “blue,” for example, allows English speakers to communicate, or signify, a representation. Upon hearing this word, a human might think of an image of blue sky, or her grandmother’s blue porcelain tea set, or of feeling “blue,” feeling melancholic. This word refers to a symbolic representation that is not a property of the word “blue” itself. Signs and symbols allow humans to communicate and pass information in a meaningful way.

As symbolic thinkers, we employ various technologies, such as language, music, art, marks on a page, which allow us to communicate through symbolic representation. Similarly, in

\(^1\)Semioticians study the role of signs and signification in meaning-making. I have drawn loosely from the field of semiotics in my research, but the rich relationship between signification and computation has only recently attracted critical attention.
machinic computation we implement these meaning-making abstractions in algorithmic form in order to automate them. But even if our "hand-wrought" computation trails automated efforts by orders of magnitude, computation – intrinsic to meaning making – remains a quintessentially human project. Why, then, do we invariably see "computation" as necessarily dehumanizing? Why has computation become, for many, the exclusive domain of science, technology, mathematics and engineering? If meaning-making through abstraction, abduction, and expression is a common feature of human experience, why do we view computation as alien and extrinsic?

Bits, or binary digits, represent the information in computation – these are the abstract symbols of computation. These bits can represent text, numbers, various symbols, and visual information. The technology might be different but the symbolic representation is common to human meaning-making and communication.

Moreover, it is a socio-historical accident that computing has been narrowed recently to involve only certain disciplines. If we accept that the process of symbolic
representation has been intrinsic to human thought since the emergence of Homo sapiens, and that machinic computation is roughly the further abstraction and automation of that process, we see that the common trope of dehumanization deserves to be turned on its head: In other words, it is not that computers threaten to drain us of our humanity, but that computers may be understood as extending and amplifying that most human part of ourselves.

Historian Michael Mahoney returns the focus of computation to the human, and the “communities of computing” that have shaped the computer’s development:

What kinds of computers we have designed since 1945, and what kinds of programs we have written for them, reflect not so much the nature of the computer as the purposes and aspirations of the communities who guided those designs and wrote those programs (Mahoney, 2005).

Mahoney argues that a combination of social, technical, political and economic factors have led to our current computational model, but computation is open to anyone who wants to learn it. It is a historical accident that “we in the humanities have had to borrow our computing from other communities” (Mahoney, 2005).
If we accept that universally, conceptual and symbolic modeling underlie many human expressions, and that machinic computation is an automation of this type of thinking, we can argue for an opening of the field to other disciplines. In Mahoney’s historical account, we can see that computation has been artificially narrowed by the communities that first embraced the technology – a mere historical accident. Symbolic representation is a collective human intellectual property. By moving away from the machine as the focus, we can make clear the possibilities for all symbolic thinkers to think computationally. One way educators are hoping to open computing to more and younger learners is through the adoption of computational thinking curriculum in K-12 and beyond.

Popularized by Jeanette Wing in 2006, Computational Thinking (CT) is a current buzz-phrase in Computer Science educational discourse, but often loosely or circularly

---

2 I will refer to the current “communities of computing” as “STEM” focused – I am addressing the popular characterization of computing as belonging to Science, Technology, Engineering and Mathematic circles. While it is not the focus of my thesis, it should be observed that a fifth, often invisible process informs the familiar four: Captialism.
defined in the literature. The Center for Computational Thinking at Carnegie Melon, where Wing made her career prior to serving as Corporate Vice President for Microsoft Research, for example, says:

Computational thinking is a way of solving problems, designing systems, and understanding human behavior that draws on concepts fundamental to computer science. To flourish in today's world, computational thinking has to be a fundamental part of the way people think and understand the world (Center for Computational Thinking, 2016).³

The above characterization of the uses or applications of computational thinking is reasonable, but acknowledges that it “draws on concepts fundamental to computer science.” If computational thinking is not determined by Computer Science itself, but draws conceptually on ideas “fundamental” to Computer Science, what is computational thinking at its basic level? What does computational thinking offer in the way of solving problems, designing systems, and understanding human behavior? Clearly, we require a more substantial consideration of this concept.

In the CT literature, Wing expands on her earlier definition, and extends it thus:

³ Author’s emphasis.
Computational thinking is a kind of analytical thinking. It shares with **mathematical thinking** in the general ways in which we might approach solving a problem. It shares with **engineering thinking** in the general ways in which we might approach designing and evaluating a large, complex system that operates within the constraints of the real world. It shares with **scientific thinking** in the general ways in which we might approach understanding computability, intelligence, the mind and human behavior (Wing, 2008).⁴

Drawing on math, engineering, and science as parallel domains, Wing argues for Computational Thinking as a form of analytical thought. Here again, though, the definition threatens to collapse under the sheer weight of inclusion: While it remains noncommittal about ways in which the humanities, for example, might also bear some parallels to the experience of Computational Thinking, her reference to the disciplines of Mathematics, Engineering, and Science are a reminder that the bits and atoms of computer thinking might be found across a wide range of human cognitive strategies. In my research, I see an opportunity to make explicit the relationship between computational thinking and activities firmly outside Wing’s scope of interest.

⁴ Author’s emphasis.
The understanding of computation in these definitions narrows the type of problem solving to a discrete set of disciplines. The opportunity lies in dissolving these borders, so that computational thinking can be conceptualized as a kind of analytical thinking for all abstract and symbolic thinking.

Paradoxically, Wing cites Alan Bundy: “Computational thinking is influencing research in nearly all disciplines, both in the sciences and the humanities” (Wing, 2008 from Bundy, 2007). Bundy makes clear that mathematicians, engineers, and scientists are not the only thinkers capable of computational thinking, nor are they the only ones for whom it is a useful strategy. But even here, the implication is that the influence of "computational thinking" on contemporary academic work is somehow unprecedented or novel. This fundamentally misunderstands human cognition.

Perhaps what we need instead of these narrowed definitions of CT, is a return to the first origins and framing of computation as an intellectual tool, an analytical strategy. Seymour Papert, the celebrated MIT professor
whose work on childhood development and computation lead to
his invention of the LOGO programming language, is a precursor to Wing and others. As early as 1980 with the
publishing of *Mindstorms: Children, Computers and Powerful Ideas*, Papert lays out his design for technology-enabled
learning environments for children. Papert, a protégé of Piaget, and constructivist theories of development, explores how computers affect the way people learn and think, especially with application to the development of children.

Given my background as a mathematician and Piagetian psychologist, I naturally became most interested in
the kinds of computational models that might lead me to better thinking about powerful developmental
processes: the acquisition of spatial thinking and the ability to deal with size and quantity. The rival approaches – deductive and knowledge based – tended to address performance of a given intellectual system whose structure, if not whose content, remained static. The kind of developmental questions I was interested in needed a dynamic model for how intellectual structures themselves could come into being and change. I believe that these are the kind of models that are most relevant to education (Papert, 1980).

---

5 Jean Piaget, a mid-twentieth century Swiss scholar whose work on the relationship between play and child development with the philosophy of Dewey, popularized constructivism: Constructivism is the viewpoint that learners build knowledge from interaction in experiential ways, with previous knowledge. Papert adds to the constructivist argument that the construction of a meaningful product is vital to this development in children (see *Mindstorms*, 1980)
Papert re-centers the emphasis of CT on the powerful computational (human and machinic) models that enhance intellectual structures and development. Papert re-focuses on the human interaction with machinic computation, and in particular, the ability of the affordances of machinic computation to bring about emergent ideas, or help elucidate ideas. Papert avoids the trap of many of the Computer Scientists above, by thinking outside of academic discipline, to the nature of learning and meaning-making in human development. He avoids the claim that CT is merely about problem solving:

Our philosophy, both implicit and explicit, tries to avoid the two common traps: commitment to technological inevitability and commitment to strategies of incremental change. I am talking about a revolution in ideas that is no more reducible to technologies than physics and molecular biology are reducible to the technological tools used in the laboratories or poetry to the printing press. In my vision, technology has two roles. One is heuristic: The computer presence has catalyzed the emergence of ideas. The other is instrumental: The computer will carry ideas into a world larger than the research centers where they have incubated up to now (Paper, 1980).

Compared to Jeannette Wing, what Seymour Papert offers us, way back in 1980, is a re-centering of the development of the mind and the intellect as the central structure of this process. It is not concepts foundational to computer
science that we are hoping to enhance with computational processes, but human thinking and meaning making.

Consider, for example, the invisible, human-computational processes that are perfectly visible to mathematicians. I see a square on a piece of paper and I recognize it plainly as a square – as an ideal polygon of perfect proportion. Papert's approach is to ask students to think in more mathematical-procedural terms.

One of Papert's most compelling arguments that he makes for childhood development is the ability to translate this spatial awareness, and object representation to early learners, through technical mediation. The ability to translate spatial awareness through computation is an affordance of technology, through a user interface. This sort of spatial development is part of the core building blocks of our intellect.

And so, without overvaluing the historical-technological particulars that have distracted us for several decades, computational thinking (CT) can now more generously be re-
conceptualized as a kind of analytical strategy characteristic of abstract and symbolic thinking.

In order to explore an application of CT, in light of this reconceptualized framing, I formulate a technical methodology in this thesis that uses computational methods (both human and machinic) to communicate meaning and abstractions through making. Through the making of patterns and textiles, and the making of quilts, I explore how abstractions and symbolic representation can be represented with and without machinic computation. In making, I design and interact viscerally with computation the fabrics themselves through programming. I explore how those algorithmic processes translate to the folk art of quilt making.

As the manipulation of signs and sign systems, computational thinking can be made visible in various ways. In the following section, I show how computational thought, interpreted, can inform the process of making even as, encoded, it serves as the subject of the process itself. In showing this, I hope to hint at the far larger, far richer world of computation beyond mere computer
science. Displaying coded patterns in quilted form illustrates one way we might think creatively about computing, and how we might encourage those outside of traditional computer science to reflect on the role computational thinking plays in their own discourse.
II. SYMBOLIC REPRESENTATION

If you supply a quilter a finished quilt, she can *reverse engineer* it. Quilters see the layers or the *stack* of component parts that create the whole: The blocked fabric topper, the cushion of mid to heavy-weight batting, and the backing fabric, stitched together in order to fuse the units. In inspecting and dismantling this stack, quilters recognize the 1/4-inch seaming of various sizes and shapes of patterned fabric, measured and cut precisely to a block
design, iterated any number of times, and pieced together to reform as whole. Quilters can feel the carefully iron-pressed and nested seams internal to the stack. Quilters can trace the quilted top stitching from corner to corner, recognizing technique and tradition, from simple cross-hatching to intricate feathering, McTavishing, echoing, and stippling. Quilters can feel age in a textile, softened by time, use, and machine washing.

A quilter is very much aware of the long and combinatorial history of quilting. A quilter most likely has a close friend or family member from whom they learned tradition and technique. A quilter recognizes terms such as "log cabin," "Ohio star," and "flying geese" as quilt blocks or folk prototypes.

It is from our shared representations, methods, and histories that the word "quilt" has come to refer to the yield of a particular bundle of processes. Any specific

---

6 Techniques for the quilt stitching which bind the layers vary vastly. These are a few of the most popular techniques, from linear (cross hatching) to meandering free form (stippling).
7 Prototype is used here in reference to the folk styles and traditions of quilting, and not to the computational use of prototype, as in prototype-based programming.
instantiation of these processes, like the quilt that rests upon my bed, stands as one possible outcome among many, as the result of a lengthy series of rule-bound and often recursive choices. In the following section, I describe those processes in terms drawn from a contemporary programming paradigm called Object-Oriented Programming (OOP), an approach to writing code that has been popular in computer science for several decades. Broadly speaking, the terms provide a useful (and rigorously defined) vocabulary for talking about how things outside of the scope of computer science – things like quilts, but also barns and poetry – were the products of computational thought before digital computers even existed.

The shared symbolic representation Quilt can be described as the class. An instance of the class Quilt, such as the quilt on my bed, is the object. The class is the template definition of characteristics that make up an object’s unique properties and methods. The object (instance of a quilt) has particular properties (characteristics) and methods (functions) within the class Quilt. An alternative instance of a quilt, a quilt object, such as the quilt in my living room, might have different properties and methods.
from the object on my bed. I borrow these terms, *class* and *object*, from Computer Science, from OOP.

Daniel Shiffman, of the Processing Foundation, describes this terminology in relation to the class and object *human*:

Let's map out the data and functions for a very simple human object:

**Human data**
- Height.
- Weight.
- Gender.
- Eye color.
- Hair color.

**Human functions**
- Sleep.
- Wake up.
- Eat.
- Ride some form of transportation.

Now, before we get too much further, we need to embark on a brief metaphysical digression. The above structure is not a human being itself; it simply describes the idea, or the concept, behind a human being. It describes what it is to be human. To be human is to have height, hair, to sleep, to eat, and so on. This is a crucial distinction for programming objects. This human being template is known as a *class*. A *class* is different from an *object*. You are an object. I am an object. That guy on the subway is an object. Albert Einstein is an object. We are all people, real world instances of the idea of a human being (Shiffman, 2016).
The concept of having a template representation, or a blueprint, and specific instantiations of that symbolic representation, are not new, however. Computer Science has borrowed these concepts from semioticians, philosophers, and linguists in order to program meaning with computational methods. Making clear this comparison is to recognize a natural process. We recombine and play with concepts across disciplines as a matter of understanding and interpreting the world that we live in. In the physical world, we use template representations, or classes, to make sense of the world. In computational object-oriented programming, we can use these same meaning making strategies. In playing with the concepts of Class and Object in reference to a quilt, I make connections to these meaning-making constructs and representations.

The instance of a quilt on my bed is built from the symbolic abstractions that are shared among human symbolic thinkers. These symbolic abstractions are part of our human-meaning-making. The instance of a quilt is the assemblage or output of complex and generative symbolic processes that refer to the past, present, and future of the symbolic representation of a Quilt. It is because of
these shared symbolic symbols and grammars of quilt making that all of the various instances are referred to as quilts, while at the same time leaving open the generatively of meaning-making – new, never before instantiated quilts can come into being.⁸

As humans we refer to shared concepts in order to make sense of the world. When these abstractions are programmed to be automated by a computational machine, we must compile the crucial properties and methods that allow us to symbolize the real-world instance we are creating. The properties and methods of a class in the real-world is not limited to an algorithm, but can be described similarly (see quilt pseudocode, Section V).

---

⁸ Generativity in language is our ability to produce and understand an unlimited set of new expressions.
III. COMPUTING AND COMPUTATIONAL THINKING

Important to the discussion of Computational Thinking, is a first a discussion of computation itself.

Computation is the automation of abstractions. Computers cannot create (think, design, etc.) in symbolic abstractions like we can, but are programmed to automate human-input in the form of abstractions. Computation involves methods that are ordered algorithmically, to transform symbolic representations, or content, in some information process. Importantly, as I’ve detailed, this sort of abstraction and expression is not the marker of computation or Computer Science, but integral to human symbolic thinking. For example, in quilt making, we have a set of ordered steps or methods, and the “stuff” to represent or encode in the process of making. We employ similar thinking with our without the machine.

It is important and has been since the adoption of the personal computer and the growth of information technologies, to understand the cognitive power of
computing to our human development. There are many forces at play that are working against integrative and inclusive learning environments in computing, including many economic and social factors. Corporate black-boxing of hardware and software, coupled with decades old biases and ageist narratives of who computing is for continue to permeate the field.

Computing is about “information processes” – a topic just as vital for the humanities as for engineering, science and mathematics. Computers are our human innovation, our tool, our foundational resource. Computing is vital to our economy, to our jobs, to our nation’s security, and to our day-to-day life. We outsource much of our thinking to computers. Just merely living along side these black-boxed technologies is not sufficient.

The idea of computation has changed. The term “computer” was first used to describe a human person performing calculations, as in “one who computes.” As technical innovations developed, calculations grew to be automated, and thus the point of reference changed as the machine became the “computer.” Computing is now more than just
machine, much more than hardware, or software. Computing broadly incorporates the study of information processes, networks, and human interactions. Peter Denning proposes a new definition of computation that captures the amorphous “thing” that is “computation” and reorients the focus from “computers” to “information representations”:

The definition proposed here [of computation] refocuses from computers to information representations. It holds that representations are more fundamental than computers...[This definition] relinquishes the early idea that “computer science is the study of phenomena surrounding computers” and returns to “computer science is the study of information processes”. Computers are a means to implement some information processes. But not all information processes are implemented by computers e.g., DNA translation, quantum information, optimal methods for continuous systems. Getting computers out of the central focus may seem hard but is natural. Dijkstra once said: “Computer science is no more about computers than astronomy is about telescopes” (Denning, 2012).

By returning the focus of computation to *information representations*, Denning is making clear that representations are the fundamental and core framework of computation. Computers automate an information process by storing, transforming and transmitting information. In Denning’s *Great Principles of Computing*, he frames *information* as “meaningful patterns (codes) of signals and states of matter” (Denning, 2015). There are other
information processes that are not implemented by computers such as DNA translation, to cite one of Denning’s examples. It is important to keep in mind Denning’s emphasis on the information process over the machine, to make clear that computation is open all complex problems that can be framed symbolically. The information processing power of computation is not limited to a particular discipline or practice.

Similarly, moving beyond the “machine” within computing, the push toward “computational thinking,” as a core foundational skill set, like arithmetic, reading and writing, thinks beyond the black box on our desk. While Wing and others who advocate for computational thinking are doing great work in opening up computation to more learners, my desire, with this thesis, is to emphasize that part of the project of “opening-up” computation is disentangling it from the mistaken assumption prevalent even among academics that “computation” lies exclusively within the domain of computer science.

These ideas and ways of thinking are not new. In taking an object such as a quilt and reverse engineering it as a
symbolic representation, I detail how similar symbolic abstractions make up this world. In embracing the push for computational thinking, I am making clear that these concepts are not foreign to humanity. We are already symbolically driven thinkers, and our world is full of abstractions.

In recognizing computation as a very human form of information processing we can see that:

(1) Computation is always the product of human choices;
(2) Computation is built of human symbolic representation;

Whether creating a quilt design, or plotting missile trajectories, technology and computation are driven by symbolically representations delivered by human agents. Technology is only "dehumanizing" if humans make it so. We can use steel to build shelters for the needy, or create weapons to kill. It is not technology that is dehumanizing, but humanity’s approach to technology that can be dehumanizing.
Given these considerations, we can view computation as an extension of human symbolic expression. We can address some of the technophobic discourse by making clear the origin of symbolic representation as that of humankind, not of machines. Automation of abstractions through computation is a progression of human symbolic thinking through technology. In the same way we have a very human concept, like a Quilt, that is built from commands, functions, methods and procedures — what we call the symbolically “coded” abstractions — we can build and automate symbolic abstractions through computing as an extension of human thinking. In the end, it is still the humans who are doing the thinking and abstracting.

In acknowledging the symbolic expressive power of computation, we can bring forward the Papertian tradition of the intellect being the center of our focus when we discuss Computational Thinking. CT offers us a vocabulary for identifying the processes whereby we conjure, shape, and interpret symbolic representation, much in the vein of semiotics. More than this, however, CT offers us a way of recalling that the serial execution of procedural statements by a digital computer, while vastly simplified,
often recalls the human processes of sense-making upon which that technology was modeled.
IV. MAKING AS A FORM OF INQUIRY

In my approach, I design and create patterns through computational methods, and then have the outputs printed on cotton by Spoonflower.¹ I transform the digitally coded textiles into quilts as a way of tackling and understanding the symbolic representations that underlie human thinking, both with and without computational mediation. Our world is becoming more connected through computation and information technologies – I welcome all efforts to create inclusive environments for all learners.

By choosing making as the form of my inquiry, I directly engage with the material and computational methods I discuss herein. I keep Papert’s focus on the intellect at the forefront of my investigation, redefining computational thinking as an analytical strategy characteristic of most abstract and symbolic thinking. Papert’s concern is building intellectual structures, whereas Wing and others seem to focus on problem solving, within the framework of Computer Science and its sister disciplines.

¹ Spoonflower.com will print and ship your uploaded designs on cotton, linen, silk, etc.
This process of quilt making, as I reverse engineered at the start, is built on abstraction, tradition, and inherited meaning. The entire quilt-object once complete is coded, two-fold — both by the computational outputs printed on cotton, and by the inherited abstractions I have absorbed from other symbolic thinkers, instantiated in a final quilt form. Through this process I am thinking computationally, but also abstractly and symbolically, as we always have. This is how we understand the world.

We abstract symbolically when writing an algorithm — a step-by-step list of instructions is written in order to produce a computational output. If a language is a means of expressing a thing — information, for one — a programming language is a means of expressing things to a computer through a set of instructions. A programming language delivers a set of instructions, or algorithm, to be read by the machine or human, to be automated. The goal of said instructions can be anything a human thinks up — the calculation of a complex math problem, the design of a software program, or the design of a digital pattern of crisscrossing lines or polka dots.
In this inquiry, I employ the Processing language and software to design and output visual designs through computational approaches. The environment of inquiry is not the important takeaway, but the idea of opening computational thinking and learning to focus on the symbolic representation of material in its many forms—whether computationally mediated or stitched into a quilt.

I chose Processing as my language of inquiry because of many of the affordances of the software and the ability to learn to code with a specific emphasis on the visual arts. Processing is an open source programming language and sketchbook, based on Java. Because it is open source, it is available at no cost. Ben Fry and Casey Reas, the developers of Processing, lay out these ideals in the Overview of the environment:

Processing is a simple programming environment that was created to make it easier to develop visually oriented applications with an emphasis on animation and providing users with instant feedback through interaction. The developers wanted a means to “sketch” ideas in code...Processing is based on Java, but because program elements in Processing are fairly simple, you can learn to use it even if you don’t know any Java.

Processing consists of:
• The Processing Development Environment (PDE). This is the software that runs when you double-click the
Processing icon. The PDE is an Integrated Development Environment (IDE) with a minimalist set of features designed as a simple introduction to programming or for testing one-off ideas.

- A collection of functions (also referred to as commands or methods) that make up the “core” programming interface, or API, as well as several libraries that support more advanced features such as sending data over a network, reading live images from a webcam, and saving complex imagery in PDF format.
- A language syntax, identical to Java but with a few modifications.
- An active online community, based at http://processing.org (Fry and Reas, 2016)

The simplicity of the Processing language lies in its use of high-level human readable language – “functions” or procedures within the language such as “ellipse” and “rect” (rectangle) are easily understood by beginners. As an example, the following Processing code instructs the computer to place a circle in the middle of the sketchbook output window:
Processing is built on an x-y coordinate plane, where the top left corner is the origin point of 0,0. In Figure 4.1, we are placing the ellipse at half the height and width of the canvas, or the center point (150,150) with an ellipse width and height of 100.

To reiterate Papert, here, most of us see a circle and think OK that’s a circle. In drawing and forming a circle through computation, we are closer to the mathematical representation of a circle, which is helpful in understanding the abstraction of phenomena in the world. A circle is a shape where all points are equidistant from the center.
The programmer can further add design features to add color, additional shapes, layers, and so on. In Figure 4.2, the background is filled in green, and the ellipse is filled in pink, with RGB\textsuperscript{10} values. The size of the ellipse is also manipulated by changing the ellipse width and height to 200.

![Screenshot of code and ellipse](image)

Figure 4.2 – simpleEllipseColor (screenshot by Olivia Payne)

Finally, in this last example, Figure 4.3, the designer can create a repeat pattern of a shape, through the implementation of a loop.

---

\textsuperscript{10} “RGB” color is a digital construction of color based on red, green and blue; this differs from the primary colors in traditional painting practice, of red, blue and yellow.
The syntax of the loop is:

```java
for (initialization; test; update) {
    statements or actions
}
```

In the draw loop of Figure 4.3, we can see that the loops initialize at y=0 and x=0, and the test are evaluative of the width and height of the canvas size, respectively. As long as the test of “y < height + 15” evaluates true, the loop will continue to run. The update adds a value each time the loop iterates; eventually the test will evaluate false. This is one way we can output equally spaced ellipses across our canvas, through algorithmic thinking.
The above example shows the simplicity of visual learning through pattern design. It shows the repetition of an abstraction, through iteration. The concept of repetition through iteration is foundational to computer programming and computational thinking. Iteration and repetition are not unique to computation, but in fact foundational to information processing in general. Recall from Peter Denning that information is defined as meaningful patterns of signals and states of matter (Denning, 2015). Similar meaningful patterns, abstracted as an example of iterative code above, are found in nature, science, and art.

In the introduction to this thesis, I detail the symbolic abstractions that make up the concept we call a quilt. In the example, I show that by repeating a block design, we decompose a greater problem (the creation of a quilt), into component parts that are more easily tackled. We represent information in these blocks that make up the whole, and through pattern recognition, build in repeated steps in order to make the process more efficient.

In programming languages, the syntax can appear foreign with the repetitive use of odd punctuation markers – “{“
and "}" or curly brackets, which signify the beginning and end of a statement block, as well as semicolon which functions to terminate a statement line; however, these simply serve as demarcating punctuation in the code. It takes some getting used to, but these are easy to adopt. Like quilting, programming languages have rules to be learned, including a specific grammars and syntax. In quilting, it is important to use a 1/4-inch seam to stitch blocks together. It is important to quilt the entire stack in close intervals as to increase the puckering of the fabric, to achieve the quilted look.

In the appendix, I lay out four additional examples of Processing code that demonstrate the pattern making possible through the computation. These examples can be used to inquire further into the computational side of symbolic representation, and to see some examples of how I’ve used making as my form of inquiry. I also include pictures of the quilts I have produced, and offer some quilting pseudocode in the next chapter in order to play with this idea of algorithmic thinking, whether through machine or human means.
V. Code / Quilt

WE MAY SAY MOST APTLY, THAT THE ANALYTICAL ENGINE WEAVES ALGEBRAICAL PATTERNS JUST AS THE JACQUARD-LOOM WEAVES FLOWERS AND LEAVES.

–ADA LOVELACE (NOTE A, PAGE 42)

Crafting and making have a long history, but the idea of hands on learning and tinkering is having a bit of a renaissance. So-called “women’s crafts” like embroidery, quilting and needle point, to name a few, have story telling power that is uniquely feminine, in the most empowering sense of the word. Women in quilting histories took scraps of fabric to make family members warm with art and form in mind. These pieces are symbolic and hold their own unique expressive capacity in the continuum of human history.

In some ways, investigating the idea of quilting on one hand, and our role in computation on the other, seem like disparate subjects – in the making of these objects, however, I am able to look at histories from a different perspective. I am able to look at computation from a different perspective. I am able to invite others who feel
removed or outside of current computational methods into thinking computationally.

The “diminutive little old ladies” who wove fibers crucial to the Apollo missions were in fact expert seamstresses, driven by a long history of women (Mindell, 2011). By downplaying their role in the getting astronauts into space we are privileging one type of tinkering or making over another. These women wove copper wires through donut-shaped cores – woven through a core was a 1, and going around a core was a 0.
Similarly, while Ada Lovelace’s intellectual curiosity with the Jacquard Loom, a weaving machine that could be programmed with punch-card technology, was vitally important to her work with Charles Babbage on the Analytical Engine (Hammerman, et al., 2015), the Loom is a mere anecdote in many of the histories of computation. The Jacquard Loom is cited as a precursor or inspiration for punch-card computation, as the loom automated complex woven pattern production. While it did not perform computation, the Loom inspired Babbage and Lovelace to design a mechanical computational device built on punch-card technology.

This techno-determinist way of thinking about the evolution of computation places the importance on the machine of today. Instead we are reframing computation as an information process of symbolic actions. I would warn against creating hierarchies of symbolic representation and meaning making – the Loom is just as vital to the histories of textile production and industrial processing, as it is inspiration for modern machinic computing. The symbolic
cognition that forms our intellect is the most vital component of all.

Finally, by reclaiming crafting and making, I offer it as an alternative to other forms of computational inquiry. In displaying my quilts and discussing these histories within this framework of symbolic representation, I hope to acknowledge these unique stories that do not often make their way into machine-oriented retellings of computation. Specifically, it is important to identify a sociohistorical account of our development, and bring common anecdotes out of a computational account of things, toward a symbolically oriented, humanistic one.

To return to the quilt, and play with this idea of abstraction, I offer a quilting pseudocode. Pseudocode is a human-readable description of the steps of a program that borrows from programming languages in its structural conventions. The program “makeQuilt” relays one set of steps I endeavor in creating a quilt from the coded textiles:
PROGRAM makeQuilt:

Design patterns in Processing;
Save png pattern outputs in Processing folder;
Upload png patterns to Spoonflower.com;

IF (printing on cotton)
    THEN select cotton;
    ELSE select linen;
Wait for Spoonflower.com to print & ship fabrics;

IF (package on doorstep)
    THEN open package!;
    ELSE do nothing;
Wash and iron fabrics;

Gather quilting supplies (sewing machine, straight pins, seam ripper, thread, rotary cutter and mat, ruler, iron and ironing board);

Design a quilt layout;

Cut fabrics into pieces;
Place pieces to sew with the patterns facing each other and press with iron;

Create quilt blocks by sewing fabric pieces together with a straight stitch on your sewing machine using a 1/4-seam;
Sew quilt blocks based on quilt pattern, row by row;
Press the seams along each row with an iron to flatten;
Sew the rows of blocks together using a 1/4-seam;

IF (all rows seamed together)
    THEN Press quilt front flat with an iron;
    ELSE continue seaming rows;

Gather the remaining quilting supplies (batting and backing);
Cut batting and backing 3 inches larger than quilt front;
Lay the backing flat on work surface;
Lay the batting on top of backing;
Lay the quilt front on top of batting;
Pin quilt radiating outward from center of quilt front;

Sew the backing, batting and quilt front together using sewing machine starting in the center of the quilt;

IF (seams stitched every 4 inches)
THEN Remove quilt from sewing machine;
ELSE continue sewing backing, batting and quilt;

Cut a long strip of fabric on the bias using rotary cutter and mat for binding;

IF (separate long strips of fabric)
THEN sew together strips;
ELSE press the binding flat;

Pin binding on edges of quilt;
Sew binding on front and back of quilt;
Finish the binding corners by hand stitching to lay flat;

Wash quilt in warm water in laundry machine;
Dry quilt in dryer;

Enjoy quilt!

End.

The program “makeQuilt,” written in pseudocode, displays a simple abstraction of the quilt making process into a step-by-step list of instructions. Symbolic thinkers can employ similar steps in abstracting and automating problems both large and small. As evidenced in the pseudocode, it is helpful to break down the complex problem of “quilt” into smaller problems. Some of the pattern recognition comes
into play when steps are repeated until a task is accomplished.

When we take our rows of fabric-blocks and seam them until all the rows are pieced together we can see how the set of instructions navigates this repetition in an efficient way:

```
IF (all rows seamed together)
    THEN Press quilt front flat with an iron;
    ELSE continue seaming rows;
```

Recognizing these patterns as targets for efficiency improvements is a vital in computational thinking, and symbolic processing more broadly. This is one example that highlights how a very human process can be written in an abstract and algorithmic way. Even if a machinic computer cannot automate these steps, the procedures can be understood and performed by a human.
Individuals in the arts and humanities can think about complex problems that could be solved with computational assistance. People making quilts have been making patterns prior to any computational modeling. That said, I find the computational modeling that I use to produce pattern as a strength of my process – I reinforce mathematical concepts, such as geometry, I play with logic and structure, and I learn by doing, in a visceral way.

Moreover, I could imagine a computational pattern generator, for example, that would assist in logging the
possible patterns you could make with $x$, $y$, and $z$ amounts of fabric. In order to build computational abstractions, I would be interested in pushing the model further, as it relates to something like quilt-making.

A quilted object is a formalized one – it is based on symbolic models, abstractions, and representations. The style and form of quilt-making has a long and re-combinatorial history, bringing together tradition, culture, and practice, as well as new technique. Computation is founded on the automation of abstractions, but symbolic abstractions are not unique to computation. By approaching computational thinking and symbolic representation through pattern and textile design, I offer one example of an alternative way of thinking about these concepts, outside of the traditional venues. Computational thinking holds an analytical power; in giving practice to computationally mediated analytical strategies, I have shown that computational thought processing is built on a shared symbolic cognition, which is built on abstractions. This intellect is unique to humankind.
VI. CONCLUSIONS AND FUTURE WORK

In this paper I lay out the foundational proof-of-concept for the opening of computation redefined as information processes based on meaningful patterns of states and matter, to a broader population. Recall from the introduction that:

(1) Computation is always the product of human choices;
(2) Computation is built of human symbolic representation;

Computation, or information processing, is always the product of human choices because we humans abstract the ideas in the first place. All of human thinking is built on symbolic representation. Learning to think computationally is really learning to think the way we already do, within new frameworks, and with new affordances and constraints.

I am starting a conversation with Jeannette Wing, from her call for computational thinking methods geared toward the K–12 student, by incorporating making and computation, and bringing the Papertian tradition to the forefront: we must
not forget about the intellectual process, and how that process is generalizable to all of human cognition, rather than a select few sister disciplines. The symbolic representation of abstraction that underlies the computational process does not reflect any particular community of thought, and thus can be conceived in new ways as the communities of computing are opened to all. There is of course a place for the humanities in all of this.

In future research, I would be interested in developing a course or course materials for humanistic disciplines and for individuals who don’t think they can code. I would like to open up this process of making, both computationally, and from shared tradition, to others. Interestingly, partially because of the integration with a craft tradition, this technique appeals most strongly to older women. I hope to infiltrate the neighborhood stitch n bitch, and bring these ideas to such an underrepresented group. I would happily bring computational thinking and symbolic representation through pattern and quilt design to such an underserved community in computational circles.
To the K-12-ers, I offer an artistic and aesthetic approach, which could be translated with ideas from Papert, code.org, and others who are making the visual a primary focus of the computational learning process. I can see a seamless translation of some of this pattern making to a combined art and computing class in existing day schools.

As part of this journey, I am challenging the humanities to think about computational thinking with art, form and history in mind, as it has a special appeal to the creative, analytical world. In the end, I will continue, of course, to make quilts and engage with computation as someone firmly placed in the humanities, and not as a “Computer Scientist.” In displaying and talking about my work, including the process, I can open the discussion of computational thinking in the Papertian and maker tradition.

Finally, I am challenging all of us to see that symbolic representation is what makes us human, and that these new technologies are only a natural progression of information processing, built of our very-human symbolic representative powers.
void setup() {
  size(1200, 1200);
  background(255, 255, 255);  //manipulate background color by changing the R,G,B values
}

void draw () {
  for (int x = 0; x < width; x = x + 25) {  //this for loop continues until the test ‘x< width’ evaluates false
    stroke(121, 232, 234);
    line(x, 0, x, height);
    //line expects x-coord and y-coord of the first pt, and the x-coord and y-coord second pt
    //goal to draw a line between two points
  }

  for (int y = 0; y < height; y = y + 25) {  //this for loop continues until the test ‘y< height’ evaluates false
    line(0, y, width, y);
  }
}

void keyPressed() {
  //to save the pattern on both Macs + PCs
  if (keyCode == RETURN) {
    saveFrame("line-pattern-####.png");
  } else if (keyCode == ENTER) {
    saveFrame("line-pattern-####.png");
  }
}
Figure A.1 - basicLines (image output from basicLines code, by Olivia Payne)
float row;       //declaring a variable ‘row’ of type float
float column;    //declaring a variable ‘column’ of type float

void setup()
{
  size(800, 800);
  background(255, 255, 255);
  smooth();
  strokeWeight(5);
}

void draw()
{
  column = random(width);    //'column' will be assigned a random
  //width each time the draw function loops
  row = random(height);      //'row’ will be assigned a random height
  //each time the draw function loops

  stroke(random(255), random(255), random(255), 50);    //randomizes the color of the line - the last value is the opacity
  line(0, row, width, row);  //line expects x-coord and y-coord of the
  //first pt, and the x-coord and y-coord second pt
  //draws a line between two points the full width of the canvas
  stroke(random(255), random(255), random(255), 50);
  line(column, 0, column, height);    //draws a line between two points the full height of the canvas
}

void keyPressed() {

  //to save the pattern on both Macs + PCs
  if (keyCode == RETURN) {
    saveFrame("plaid-pattern-####.png");
  } else if (keyCode == ENTER) {
    saveFrame("plaid-pattern-####.png");
  }
}

}
Figure A.2.1 – basicPlaid 1 (image output from basicPlaid code, by Olivia Payne)

(Figures A.2.1-A.2.3 illustrate three instances of looping code; the basicPlaid program will run until it is stopped.)
Figure A.2.2 - basicPlaid 2 (image output from basicPlaid code, by Olivia Payne)
Figure A.2.3 - basicPlaid 3 (image output from basicPlaid code, by Olivia Payne)
int x;        //declaring a variable ‘x’ of type int
int y;        //declaring a variable ‘y’ of type int

void setup() {
    size (1200, 1200);

    background(66, 216, 197);

    noLoop();           //stops Processing from continuously executing
    //the code within draw()
}

void draw() {

    for (y=10; y< height; y= y + 25) {     //while ‘y< height’ holds
        true, stay in loop; when y is > height, exit loop

            for (x = 0; x < width+15; x = x + 25) {
                fill(216, 66, 71);          //adjust to change fill color of egg
                stroke(0, 0, 0);            //adjust to change color of outline
                strokeWeight(1.5);          //adjust for thicker outline

                //noStroke();              //optional - removes outline from egg

                ellipse(x, y, 20, 27);      //ellipse expects x-coord, y-coord,
                //width, height

                println(x);                 //test of the code - is it running?
            }
        }
    }

void keyPressed() {

    //to save the pattern on both Macs + PCs
    if (keyCode == RETURN) {
        saveFrame("egg-pattern-####.png");
    } else if (keyCode == ENTER) {
        saveFrame("egg-pattern-####.png");
    }
}
Figure A.3 – brokenEggs (image output from brokenEggs code by Olivia Payne)
boolean isMirrored;  //boolean allows us to turn on/off function by changing from true to false

void setup ()
{
  size(1200, 1200);
  background(94, 127, 222);

  //customize - change the background!
  //you can use Tools > Color Selector to get RGB values

  isMirrored = true;  //change to false to turn off mirroring
}

void draw () {

  //translate(mouseX, mouseY);  //will draw object where your mouse is positioned
  //rotate(frameCount*0.5);  //rotates the object around the point of origin - mouseX, mouseY
  if (mousePressed)  //if your mouse is clicked, the shape will be drawn at mouseX, mouseY
  {
    if (isMirrored) {
      mirrorDraw();
    } else {
      fill(random(200, 255), random(5, 155), random(25, 145));  //fill(Red,Green,Blue) - fills the shape with color
      //randomizes the color each time through the draw loops
      //note that the color is randomized within bounds, heavy on the RED

      translate(mouseX, mouseY);  //will draw object where your mouse is positioned
      rotate(frameCount*0.5);  //rotates the object around the point of origin - mouseX, mouseY

      bezier(0, 0, 55, 15, 15, 55, 0, 0);

      //this bezier expects:
      //anchor pt, anchor pt, control points x4, anchor pt, anchor pt
      //in this example, we want the anchor points to end and begin
      //at the origin pt - which is: mouseX, mouseY
      //you can play with these numbers to create different shapes!
    }
  }
}
void mirrorDraw() {
    fill(random(200, 255), random(5, 155), random(25, 145));
    pushMatrix();
    translate(mouseX, height-mouseY); //will draw object where your mouse is positioned
    rotate(-frameCount*0.5); //rotates the object around the point of origin - mouseX, mouseY
    bezier(0, 0, 55, 15, 15, 55, 0, 0);
    popMatrix();

    pushMatrix();
    translate(width-mouseX, mouseY); //will draw object where your mouse is positioned
    rotate(-frameCount*0.5); //rotates the object around the point of origin - mouseX, mouseY
    bezier(0, 0, 55, 15, 15, 55, 0, 0);
    popMatrix();

    pushMatrix();
    translate(mouseX, mouseY); //will draw object where your mouse is positioned
    rotate(frameCount*0.5); //rotates the object around the point of origin - mouseX, mouseY
    bezier(0, 0, 55, 15, 15, 55, 0, 0);
    popMatrix();

    pushMatrix();
    translate(width-mouseX, height-mouseY); //will draw object where your mouse is positioned
    rotate(frameCount*0.5); //rotates the object around the point of origin - mouseX, mouseY
    bezier(0, 0, 55, 15, 15, 55, 0, 0);
    popMatrix();
}

void keyPressed() {
    if (key == 'c') {
        background(94, 127, 222);
        //this functions to "clear" the screen
    }

    //to save the pattern on both Macs + PCs
    if (keyCode == RETURN) {
        saveFrame("bezier-pattern-####.png");
    } else if (keyCode == ENTER) {
        saveFrame("bezier-pattern-####.png");
    }
}
Figure A.4 – bezierFlowers (image output from bezierFlowers code, by Olivia Payne)
BIBLIOGRAPHY


