WHERE'S THE REVOLUTION? FROM "CODE YEAR" TO THE CONTINUUM OF PROCEDURACY

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Christopher Aaron Lindgren

In Partial Fulfillment
for the Degree of
MASTER OF ARTS

Major Department: English
Program Name: Composition

June 2012

Fargo, North Dakota

North Dakota State University

Graduate School

Title

Where’s the Revolution? From “Code Year” to the Continuum of Proceduracy

By

Christopher Aaron Lindgren

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University’s regulations and meets the accepted standards for the degree of

MASTER OF ARTS

SUPERVISORY COMMITTEE:

Dr. Kevin Brooks
Chair

Dr. Andrew Mara

Dr. Kelly Sassi

Dr. Brian Slator

Approved by Department Chair: Dr. Kevin Brooks

06/18/2012	Kevin Brooks
Date	Signature

ABSTRACT

As the calendar turned over to 2012, an online learning initiative, Codecademy, declared it “Code Year”—the year “for everyone” to learn code. Within six months, this call has received much attention from the public and scholars in the university. Yet, this history and theory paper more deeply investigates this call for a new mass literacy, which was actually proposed back in the 1960s as procedural literacy, i.e., proceduracy. Accordingly, a history is told about how Computer Science ignored Alan Perlis’ call for procedural literacy and Rhetoric and Composition has just recently begun to address Marshall McLuhan’s media turn. From there, this paper connects new scholarship and applications surrounding the unexamined persuasive and expressive faculties of processes to literacy scholar Annette Vee’s levels of proceduracy. Finally, conclusions and implications of Rhetoric and Composition’s involvement in the deeper engagement with the writing of code are discussed.

ACKNOWLEDGEMENTS

I have many people to thank for their help during the development of this thesis. First and foremost, I am thankful for my mentor Kevin Brooks, who has always challenged my scholarship both theoretically and more importantly how such ideas live beyond the printed page or networked screen. I am thoroughly indebted to his invitations for research, which, perhaps, he may be just as thankful that I replied, "Yes" to take on such large, but important outreach, research and development projects. Thank you!

My time at NDSU has been a time of tremendous growth and experience of great support from the faculty, staff, and colleagues in the English department. Andrew "Doc" Mara has the amazing ability to push and challenge ideas in the classroom or over a quick conversation in the hallway, coupled with the knowledge on how to launch such ideas into projects beyond the class. Kelly Sassi has provided some of the most thoughtful and constructive feedback on my projects, and has taught me vital scholarly skills, methods, and organization via her own research and teaching. Betsy Birmingham inspired me with her amazing course in Language Bias, which also provided me with the opportunity to take on my first IRB-approved research project. Amy Rupiper Taggart always challenged my ideas and pedagogy, while simultaneously remained in full support of the trajectory of my work in and out of classroom. Thank you to my outside committee member, Brian Slator, who provided some valuable insights at the final leg of this project.

Thank you to my colleagues and friends, Steven Hammer and Becca Hayes, who heard more about proceduracy than they bargained for during the final stretch of this project. I would also like to extend a special thanks to Steven for supplying the much-needed support with our sound-based projects; these things kept me motivated and energized throughout this journey. I'm looking forward to seeing all of our careers unfold in the coming years.

DEDICATION

This work is dedicated to Laura. You are my friend, unwavering support and companion, who inspires all that I do in my work. Thank you for everything.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

"What then is computer literacy? It is not learning to manipulate a word processor, a spreadsheet or a modern user interface; those are paper-and-pencil skills. Computer literacy is not even learning to program. That can always be learned, in ways no more uplifting than learning grammar instead of writing." (Alan Kay, “Computer Software” 59)

Learning How to Code is Not the Revolution

Since the 1960s, Rhetoric and Composition has struggled to incorporate the visual, digital, and networked turns, which have been called revolutionary in their respective times, in both the theory and praxis of rhetoric and writing (Crowley; Sirc *Composition as a Happening*, “Resisting Entropy;” Rice; Hawk). Specifically, Jeff Rice argues that our field missed Marshall McLuhan’s challenge for us to consider how writing will change in the emerging electronic age. Since this challenge, and the many social, cultural, and educational turns within our field, we are confronted with yet another call by Codecademy to make 2012 “Code Year.”

In this thesis paper, I will neither adopt the essentialist and revolutionary driven rhetoric of the Code Year movement that seems to have pushed beyond the scope of Codecademy itself, nor stand by in this moment and ignore this call to code in our field either. Instead, I will investigate the history of proceduracy, which Literacy scholar Annette Vee defines as the ability to break down a complex problem or situation and subsequently write an algorithm out of smaller procedures to be written as code to be compiled and performed by a computer (“Proceduracy” 3). Furthermore, I will discuss its relationship to literacy, so I can provide a vision on how to position proceduracy in a way that makes sense to both fields: Computer Science and Computers and Composition/Writing – a position that will hopefully begin to encourage collaboration and mutual respect of the ways proceduracy manifests in various cultural contexts across the university.

This journey to position "Code Year" as a key moment for new inquiries into proceduracy and the nature of writing code begins with an analysis of the present day call to code by the online educational initiative, Codecademy. Through John Trimbur's "Literacy and the Discourse of Crisis," which defines every call for a new literacy as an ideological event, I will argue how the goals of the "Code Year" call are rather stagnant and in opposition to becoming a useful educational model for the academy and public writ large. Ultimately, this thesis will consider how proceduracy has the potential to create the conditions for learning to code across the curriculum, enacting new identities and cultures surrounding computation and writing code.

Code Year's Discourse of Proceduracy Crisis

Trimbur argues that "literacy crises are always strategic" (286). Such educational initiatives are typically framed through fears of downward mobility, which has repeatedly "led the middle classes to identify their private interests with an increasingly stratified and meritocratic order in education" (293). He states that any "literacy crises take place when a cultural lag occurs, when literacy practices and literacy education have not quite caught up to increased expectations and heightened demand" (284). Trimbur argues that this "cultural lag" is an "ideological event" (281), which occurs when the culture in power either frames the crises experience as a noticeable decline in ability or as a result of "heightened expectations and increased social and economic demand" of the literate skill (284). To locate the most recent “cultural lag,” I look to Codecademy’s “Code Year” campaign as the emblematic articulation of the present discourse of crisis that claims to “make it easy for everyone to love and learn how to code” (“About” 2012).

As the calendar turned over to 2012, Codecademy launched their “Code Year” campaign for people to learn “how to code this year” by supplying “interactive programming lessons” in JavaScript via email every Monday throughout the year (“Code Year”). On Codeyear’s landing page, numerous technological entrepreneurs and some scholars are cited to support their initiative, such as a Partner at Union Square Ventures, who is quoted as saying, “A young man asked for advice for ‘those who aren’t technical.’ I said he should try to get technical” (“Code Year”). Tim O’Reilly, CEO and Founder of O’Reilly Media, is quoted as saying, “I wish I'd thought of [this]...The need for computer literacy has never been greater. This is a fabulous product at the right time” (“Code Year”). Additionally, Douglas Rushkoff’s tagline driving this cultural zeitgeist, “program or be programmed,” can be seen throughout the site. What you will not find are references to proceduracy and more critical and robust ways of engaging the rationale to learn how to code in 2012. Instead, the major cultural message coming out of the “Code Year” program connects back to Trimbur’s discourse of crisis, which I would update from traditional literacy crises to the *crisis to code* that has been developing since the 1960s.

In just six months, the “Code Year” program currently has over 450,000 subscribers and has been circulating in the public sphere via academic blogs (Salter; Dwarok; Talbert) and the national press, most notably, a story about New York City Mayor Michael Bloomberg’s vow to learn how to code via a twitter message to highlight the impo (“NYC Mayor”). Over the past few years, there has also been an increase in other movements that maintain the position that more coding and/or technological gains must be made in education, (e.g., Google's "Code Camp" and "Computational Thinking" programs, emerging STEM schools, President Obama's Race to the Top program, etc.). Furthermore, the call to code in 2012 could be characterized by technopragmatist and HASTAC co-founder Cathy Davidson's call for our educational system to take up a fourth "R" in the standard Reading, wRiting, and aRithmetic to now include algoRithm to prepare the new generations for the 21st century. Yet, this idea to "get more technical" and coordinate a mass educational reform around coding has happened before.

What makes "Code Year" an updated case of Trimbur's *discourse of crisis* into a proceduracy crisis is how it demarcates the skill through the middle/upper classes call to maintain the goal toward economic security by simply "getting technical." Presently, discourse surrounding "Code Year," as well as the activities provided on the website, fail to acknowledge such work to see how learning to code is more than a professional skill rooted in a particular grammar and style. While many of these examples remain outside of the university, they emphasize the discursive loop that our culture seems to find ourselves in: Johnny can't write, and now he can't code. Here, as Trimbur's claims are validated once again, I suggest that simply learning how to code is not the revolution, because, as I will discuss in this thesis, this "Code Year" movement should have more critical goals beyond simply getting the code to compile. Instead, proceduracy folds in more of the robust rhetorical traditions that our field can offer to the writing of code. Furthermore, I will provide a history surrounding past calls to code in both Computer Science and Rhetoric and Composition that have always stressed To begin this journey toward a more critical engagement with Code Year, I submit that a good place to prime the conversation is via the

"Program or be programmed" town hall meeting at the 2012 Computers and Writing conference, which also recognized this zeitgeist surrounding Codecademy's "Code Year" campaign.

As Computers and Writing entered the discourse of this coding crisis, many of the town hall's panelists pushed for many in this subfield to learn how to write code. Rhetoric and Digital Media scholar David Reider stressed that learning how to write algorithmically will enable us to see how words come alive via code or we risk falling behind the curve. Literacy scholar Annette Vee discussed the values behind coding practices and how groups define what "good" code is to emphasize that "there is no good code" without the recognition of its source social context ("Coding Values"). Literary and Media Studies scholar Mark Sample cautioned the field to not use the word "literacy," as it enacts this "program or be programmed" rhetoric. He instead suggested using the term competency to emphasize the "highly contextualized, situated, and fluid" nature behind learning a literate skill." Karl Stolley, a digital rhetoric and writing scholar who is also a web developer, "crafted" his vision toward "source literacy" that stresses the work to become more comfortable at the keyboard to foster a "command-line literacy." Out of each of these panelists, I am most closely aligned with Vee, whose work I will come back to in Chapter 3.

Rather than simply urge our field to take up programming, I will first go back to the early 1960s in chapter 2, to provide a parallel history between the newly formed disciplines of Computer Science and Rhetoric and Composition to show how both disciplines overlooked the initial calls to the revolutionary proposition for procedural literacy by Computer Scientist Alan Perlis and media scholar Marshall McLuhan's challenge of the university to engage the emerging electric age. The goals for this history will not be to draw lines in the proverbial sand between who needs to address what aspect of proceduracy, but instead I hope it will elicit a mutual respect for each others roles in the re-call for proceduracy across the disciplines, where I submit that those who are committed in our field of Rhetoric and Composition will use Code Year as a means to research and develop scholarship surrounding the rhetorical faculties of code. In chapter 3, I review the contemporary calls and theories on proceduracy, which I will extend with the examples from interdisciplinary ideas and goals from fields such as videogame theory, media theory, and software studies. Finally, in chapter 4, I will provide some of the implications of these ideas on distributing proceduracy across the disciplines, and, more specifically, how Rhetoric and Composition should pursue this line of work.

CHAPTER 2. THE MISSING HISTORY OF PROCEDURACY

In this chapter, I will focus on converging the histories of both Computer Science and Rhetoric and Composition as newly formed disciplines in the early 1960s to highlight their shared experiences in missing the calls to learn from the critique and writing of models throughout the university. Rhetoric and Composition could have taken up McLuhan's call to explore and teach networked thinking in the electric age, but instead taught composition via the rearview rhetorical tradition. Interestingly, McLuhan's challenge to Rhetoric and Composition on the effects of the electric age on writing has had a strong influence on one of the key figures in Computer Science, Alan Kay, but, as this history reveals, even Kay is still telling us all that "The revolution hasn't happened yet."

I will make no attempt to provide a full historical account of either discipline's relationship with software development and use or situate each account intellectually. Instead, I chose to understand the effects of missing the opportunity to engage with computers and software in more diverse and critical ways via procedural literacy; particularly in relation to the ongoing research and praxis of what writing is and how it operates in the world.

Computer Science Overlooks Procedural Literacy

Back in 1961, at a symposium during the 100th anniversary celebration of M.I.T., software engineer Alan Perlis proposed that computer programming embodied a variety of skills that are relevant and useful to numerous disciplines in the university. In "Procedural Literacy," new media scholar Michael Mateas describes Perlis' call as a revolutionary idea that sought to teach computation and complex processes across the disciplines by starting an introductory course to the university for every incoming student. Perlis' response drew from his observations that recognized the patterns of typical exigencies on how and why people interacted with computers and the learning of a specific programming languages (Mateas 103). He observed that:

> most students learn to use computers in relatively haphazard ways, either driven by the need of some particular application, or in the context of a numerical analysis course that is primarily focused on teaching numerical methods, or on their own, or in a course that teaches some particular programming language. None of these approaches focus on the teaching of computation. (103)

As aforementioned, Perlis attempted to move away from the model of teaching a specific language, and, instead, desired to teach computation to all incoming first-year students and beyond at the university. Teaching computation, to Perlis, is how students are able "to construct complex processes out of simple ones—what would become labeled as *procedural literacy* (qtd. in Mateas 105). To approach computational thinking, he outlined the goals for a set of courses across the disciplines that would teach programming as a medium designed for the ability to describe and write processes, which, according to Perlis, aligned with the ultimate goals of any university. Mateas summarizes these goals for students to be taught the following principles:

1. A cultural grounding for knowledge: "sensitivity… a feeling for the meaning and relevance of facts" (Perlis qtd. in Mateas 103-104)
2. To teach students how to think about and communicate models, structures and ideas: "…fluency in the definition, manipulation, and communication of convenient structures, experience and ability in choosing representations for

the study of models, and self-assurance in the ability to work with large systems…" (104)

3. To teach students how to educate themselves by tapping the huge cultural reserves of knowledge: "…gaining access to a catalog of facts and problems that give meaning and physical reference to each man's [sic] concept of, and role in, society." (104)

Considering his emphasis on cultural and communication fluencies, Perlis' goals align with the goals of most Rhetoric and Composition educators and are arguably more ambitious than the goals of most writing or computer science programs presently. Based on these goals, Perlis desired to create the conditions for students to enter the university to "develop an intuition for which problems and ideas are important or relevant" (Mateas 103). As a result, students must both engage with technology more critically, while also emphasizing the social nature of learning. Mateas interprets these guidelines and sees the "procedurally literate practitioner" as a person who is "aware of the constraints of specific tools, [and] will be capable of considering a space of computational possibility larger than any specific tool" (105).

Despite Perlis's suggestion for a move toward procedural literacy at this forum, other software engineers—notably, Peter Elias—opposed this piloting of a program plotted toward a general education outcome that foretold computation enacted across the disciplines. In response to Perlis, Elias posed an antithetical direction that would ultimately become the model to situate the teaching of specific programming languages within the domain of Computer Science alone. Elias's vision desired to develop and "train" their undergraduates in specific programming languages. This goal, argued Elias, would "generate enough worthwhile languages for [Computer Science] to be able to stop," meaning stop the research and development of new programming languages (qtd. in Mateas 104). Nowhere is this more evident in his description of the coming "natural" abilities of future engineers and scientists, who, in Elias' words, "will face the console with such a natural keyboard and such a natural language that there will be very little left, if anything, to the teaching of programming" (qtd. in Mateas 104). In direct opposition to procedural literacy, Elias concludes to the council that if they will do anything different as an emerging science and discipline, they will have "failed" (qtd. in Mateas 104).

Elias' vision won out, but it has not gone unquestioned or challenged. Literacy scholar Annette Vee writes that "human conversational fluency has been a holy grail for language design; all languages still require processes to be broken down, made explicit and then expressed in ways the computer can understand" (*Proceduracy* 44). Lead programmer at IBM in the 1960s Frederick Brooks wrote an influential essay, arguing that there is "no silver bullet" programming language to address every problem (Brooks qtd. in Vee 43). Mateas also questions Elias's vision for students to learn how to "speak to machines" as a natural ability by interrogating such naturalness as being synonymous with a state of becoming "frictionless" (105). He refutes Elias's modernist notion of "stopping" the evolution of programming languages, which, as history has proven, software engineers have developed thousands of programming languages and will continue to develop even more over time (105). This shortsighted goal of Computer Science negated the critical insight of how the computer was and is the new medium that is "about describing processes … [and] complex flows of cause and effect," which, as Mateas explains, "will always involve work, never achieving this frictionless ideal" (105). Ultimately, Elias' vision failed to acknowledge the semiotic domain of programming, suggesting that there will (and should) be only a few languages to develop and use, which established a trajectory for the field to disciplinize itself as a Science, focusing its efforts on language development and efficiencies, rather than understanding the complexities of proceduracy development and how it informs its instruction.

Of course, these pursuits are necessary for the advancement of computing and software design, but computer science's shouldering of the burden as the sole entity in the university to teach programming since the 1960s has beset unrealistic expectations on its departments, instructors, and students. To convey the magnitude of the issues created by keeping the role of teaching programming confined to computer science departments: since their inception in the 1960s, between 30% and 60% of every university's … intake fail the first programming course" (Dehnadi and Bornat 1). Researchers Saeed Dehnadi and Richard Bornat (Middlesex University) contextualize these statistics by indicating that "many of the [students]" who fail these courses "are mistakenly … 'progressed' into following courses" (n1, 1). They connect these procedures of the department to the "misguided Quality Assurance procedures and the efforts of colleagues who doggedly believe in the normal curve" (n1, 1). Ultimately, the introductory level courses in computer science serve as a gatekeeper to the one mode of learning how to program, which has also served as a deeply entrenched dichotomy between learning how to program across the disciplines.

Yet, instead of a "revolution" in computer use and programming, influential computer scientist and software engineer Alan Kay argues that the current model curricula in the university could not keep up with the quick developments of the technology, and the academy compounded this problem through the blackboxing of such developments in software from the rest of the institution and culture writ large. Consequently, Computer Science shouldered the entire responsibility to teach programming and the affordances of this new medium, which, as Kay argued, led to its overemphasis on professionalization. In an interview with ACM Queue, Kay elaborates on the aforementioned flawed state of learning how to program, stating "that most undergraduate degrees in computer science these days are basically Java vocational training," which alludes to the stronger cultural causes, i.e., procedures, driving these non-revolutionary uses of computers (qtd. in Feldman). He criticizes the capabilities of software and the far from revolutionary use of the networked information systems prevalent in our culture. In similar fashion to Perlis' call for procedural literacy, Kay desired that the future of educational environments and their research, development, and use of software "are not going to retrieve facts but *points of view*" (qtd. in Kasch, emphasis added).

Throughout the 1970s, Kay and many others (Dan Ingalls, Adele Goldberg, Diana Merry, etc.) developed the core essence of the personal computer of today at the influential Xerox PARC (Palo Alto Research Center), including the graphical user interface (GUI). Kay and Goldberg developed the first object-oriented language environment, Smalltalk, which was designed for anyone to be able to program their own tools and simulations ("Personal Dynamic Media"). Kay describes his vision for computers and education, akin to Perlis' goals with computation with procedural literacy, stating that:

> The weakness of databases is that they let you retrieve facts, while the strength of our culture over the past several hundred years has been our ability to take on multiple points of view. It should be possible for every kid

> everywhere to test what he or she is being told either against arguments of others or by appeal to computer simulation. The question is: will society nurture that potential or suppress it? (qtd. in Kasch)

Despite the seriousness of Kay's tone and point, at the start of this particular excerpt, Kay, in McLuhanesque fashion, says more than what he means at face value, when he states that "The weakness of databases is that they let you retrieve facts." In this 1991 interview, Kay understands that the *purpose* of databases are to retrieve facts, but by reframing it as a *weakness*, he highlights Computer Science and our culture's overemphasis on the computer as a tool, rather than a medium. Kay's question of the nurturing or suppression of this vision connects back to the misprision of computer science, where he argues the field has largely neglected the architecture of programming, and instead focused on the materials (data structures). Kay, among numerous other computer scientists, since the 1960s, has taken on this challenge to not only develop more advanced programming languages, but also expose students to the potential power of computers and computational thinking as a way toward seeing, mapping, and navigating the numerous cultural systems, (more resemblances of proceduracy), which have yet to become adopted practices and philosophies within American educational institutions.

In numerous interviews and speeches, Kay references McLuhan as one of his primary influential thinkers who has informed his outlook on computing and its role in education akin to proceduracy. More recently, in 2007, he addressed the University of Pisa in Italy about the still forestalled revolution in computing by explaining the reigning influence of the 15th century invention of the printing press, which is discussed at length in McLuhan's *The Gutenberg Galaxy*. Kay cites McLuhan's work, stating that "when a new medium comes along it is first rejected on the grounds of ‘too strange and different,’ but then is often gradually accepted if it can take on old familiar content” (Viewpoints Research Institute 3). He notes how many of the major software engineers in the 1960s, who went on to develop personal computers, bit-mapped screens, overlapping windows, … the Ethernet, and the Internet, were motivated by the highest transformational achievements of the printing press” (Viewpoints Research Institute 1). Essentially, Kay maintains that the computer is the first medium that *could* remediate all of our sound and visual senses into “yes” or “no” binary numbers. He draws repeatedly from McLuhan’s argument that "the electric age brings number back into unity with visual and auditory experience, for good or ill" (*Understanding Media* 109).

The computer is the medium best suited for people “to explore the universe of knowledge,” argues Kay (59). Since he recognized that print, as a medium, gives shape to our ephemeral thoughts, while software, as another medium, gives shape to our ways of doing and interacting with systems. He foresaw the computers ability to able to give new agency to the written word, because “the computer could carry out the implications of the claims to provide a better sense of whether the claims constituted a worthwhile model of reality” (Kay, Viewpoint Research Institute 2). In a 1984 article in *Scientific American*, Kay describes the computer as the next wave in literacy and expression, stating that:

> The protean nature of the computer is such that it can act like a machine or like a language to be shaped and exploited. It is a medium that can dynamically simulate the details of any other medium, including media that cannot exist physically. ... It is the first *metamedium*, and as such it has degrees of freedom for representation and expression never before

encountered and as yet barely investigated. (emphasis added, “Computer Software” 59)

In 1997, Kay delivered a widely cited keynote address, “The Computer Revolution Hasn’t Happened Yet,” that challenged the field of computer science to consider how “there has [yet] to be an exquisite blend between beauty and practicality” in the production of programming languages and consequently software. He laments how the field and professional industry has failed to understand how “[a]rchitecture is always going to dominate material,” and more specifically how, according to Kay, it took the advent of the Internet to even spark an interest in the architecture of Object-Oriented Programming.

Even though Elias’s vision became the model enacted in the academy, suppressing Perlis’ call for a truly revolutionary model of procedural literacy and Kay’s vision to decentralize computing across the educational sphere, Kay still continues to promote his vision. Now, during the cultural zeitgeist of Code Year, perhaps, this opens the potential to finally explore the ability and art of writing code to be explored in both theory and praxis across the disciplines; specifically in Rhetoric and Composition. Here, at this moment, we can begin making more connections between the potential repurposing and fusion of our work.

Rhetoric and Composition Overlooks the “Message” of Media

While Kay understood the revolutionary implications of the work of literary and media scholar McLuhan, the disciplines of English and Rhetoric, unfortunately, did not. Jeff Rice expanded on the effects of the electric age on Composition Studies by calling back to 1963 as a “’watershed’ moment in the history of writing, … [where] various writers simultaneously explored, … the connection between writing and culture (56). In this, yet another, call back to the 1960s, Rice emphasizes the exclusion of McLuhan's (*The Gutenberg Galaxy* 1962; *Understanding Media* 1964) challenge to "print-based thinking" in the overarching discussion during the formation and evolution of Composition Studies (56). Specifically, Rice argues that Composition has failed to consider how McLuhan's call to recognize how the "'new configurations of mechanisms and of literacy'" disrupts print-based "'forms of perception and judgment'" of what writing is and how it functions in a digital culture (McLuhan qtd. in Rice 56). Rice concludes by calling into question Composition's history, which is grounded in "a print model," and calls for a revolution in how media theory and production should inform Composition's theory and pedagogy.

This media-driven revolution is not too far removed from the dispersed models of computational/procedural literacy proposed by Perlis and practiced by Kay to write mediums with an emphasis on cultures. Rice proposes a Composition theory and pedagogy where "dominant ideologies become subjected to newly created writerly control," blurring the lines of power in such discourse practices, "between those who create discourse and those who receive it" (70). This media-focused method is an attempt to see information not as something that is delivered, rather as something that teachers and learners traverse with and manipulate—it is a call to see and code the environments in which we inhabit and produce discourses.

At first glance, Rice's argument for a media-based model of composition positions the field to simply "teach HTML, weblogs, or other new media-based assignments," but he emphasizes his concern "about the much larger issues of curriculum and pedagogical practice… [that calls] for a closer correlation between writing and media in our pedagogy and in our theorizing" (70). Considering Rice's call back to 1963, around the same moment when Computer Science neglected to take up Perlis' revolutionary proceduracy-informed curriculum, Rhetoric and Composition chose to continue its long tradition and print-based thinking over the challenges presented by the new metamedium, which would come to produce a new model of networked thinking.

What Happened Instead of the Revolution in Rhetoric and Composition?

Just as Computer Science missed the revolutionary power of computing by sidestepping the recognition of computers as environments, so, too, did the germinal scholars and educators in Computers and Writing field miss the revolution. The initial era of Computers and Writing ushered in a series of homegrown software programs that served the individual writer. In the early 1980s, Computers and Writing scholar Lisa Gerrard notes that many composition instructors were developing software, such as word processors, invention aids, and style checkers (282). Yet, the logic driving the use of the computer to guide an individual writer through the invention process, at its core, draws from the processes instantiated by the rhetorical revival era: the desire to use classic rhetoric to create the conditions for a "systematized guidance at every step in the writing process," and this is exactly what the first era of Computers and Writing achieved (Corbett qtd. in Rice 58). These programs and breadth of scholarship from the emerging subfield of Computers and Composition emphasized the use of the computer as a personal device, which foregrounds larger issues related to not envisioning the computer as a networked, social machine—an issue of inattention to the architecture that has intersections with Kay's grief with Computer Science.

The programs created by Computers and Writing scholars, such as Hugh Burns' TOPOI, Helen Schwartz's SEEN and Prewrite, Donald Ross and Lillian Bridewell's

ACCESS, Gerrard and Ruth Von Blum's WANDAH, and James Strickland's FREE were designed to instruct a well-defined, user-friendly path through the writing process. Since these programs drew from more traditional theories of rhetoric, Gerrard wrote in her review of the first decade of the Computers and Writing conference, "Computer literacy--whether applied to student or instructor, and whether it meant the ability to use a keyboard, to program, to analyze the social consequences of computers, or a combination of these—was always defined as a personal, rather than social, skill" (283). Afterall, the first computers in the homes of non-specialists were not networked machines. Throughout the 1980s, they were marketed as the next household appliance and sold as personal devices that could do what machines do best: crunch numbers. While this is, of course, true, computers have proven to contain far more potential beyond this hindsight, but as Kay notes, our educational and political institutions could not keep up with the scalability of this new medium. Consequently, however active the scholars and practitioners were in either Computer Science or Computers and Writing, the computer became the *personal* computer, (the PC), and in conjunction with the rise of commercial software, Gerrard notes, almost in passing, that "the days of working with a programmer or writing our own code [were] largely over" for scholars and practitioners in the Computers and Writing field (282).

Ironically, just as Computer and Writing/Composition's role turned away from developing software, Paul Leblanc published *Writing Teachers Writing Software* (1993). While our field has largely overlooked Leblanc's work (even within Computers and Composition), his book catalogs the experiences of the first wave of Composition scholars involved in various methods of program development for the field of writing instruction. Referring back to Gerrard's indication of the shift away from software development, Leblanc argues that this redirection originated mainly from the difficulty for tenure-track scholars to develop software, which, at the time, did not include the writing of writing tools on the computer. Procedurally, scholars in Computers and Composition understood this institutional restraint, leading to a shift toward the adoption of proprietary software in the classroom and scholarship. Consequently, as Cynthia Selfe and Dickie Selfe claimed, most composition scholars and educators, "deal[t] with technology not as critics but as users" (496), so, throughout the mid-to-late 1990s, the rise of proprietary software packages in Computers and Composition begot a new series of influential scholarship centered around "the politics" of the interface, code, and overall nature of writing on the surfaces provided to us on the web (Bolter 1991; Selfe and Selfe 1994; Johnson-Eilola 1993; Selber 1997).

Bill Hart-Davidson describes his own experience during this time period, realizing that "the tools and environments in which [he] was writing, and those in which [he] was encouraging [his] students to write, were built out of the very stuff that [he] was supposed to be in command of as the teacher: assumptions, theories, principles, and pedagogies of writing" (qtd. in Inman 200, 95). Due to this problem, the subsequent praxis and scholarship of Computers and Composition scholars called for more critical use of the technology used in both the classroom and workforce, paving the way for future scholars to cross the form/content divide.

The Computers and Composition/Writing community crossed this divide by first critiquing and writing on the surfaces of the web via hypertext, the occasional writing of script-based code (Bolter; Johnson-Eilola; Selber), and analyzing the designs of interfaces (Selfe and Selfe). Yet, this new wave of scholarship rarely pushed for writing instruction that incorporated structured programming languages in the classroom. Instead, it called for users to become better *readers* of interfaces, delineating an agency in digital mediums through the creation of new interfaces via more visual-based methods outside of code, e.g. drawing and writing with either on paper or proprietary software (Selfe and Selfe). Although, in 1999, one issue of *Computers and Composition* dedicated itself to position some Rhetoric and Composition scholars to research and praxis surrounding code.

Most applicable to my arguments here, Joel Haefner's arguments in "Politics of Code" represents the perpetuation of code as a limited language that remain prevalent in contemporary scholarship, when compared to the performativity of human language in discourses. He argued that "the language of coding is much more limited in the kinds of *actions it can perform* than the language we use for communication every day, and in writing classes (emphasis added 326). He compares code to Michel Foucault's argument that, after the Renaissance, "[l]anguage became self-referential, rule-governed, and hierarchical," implying that Modernity's "manifestation of 'the will to truth'" is embodied in the "Boolean logic" of "the electrobinary world of the computer itself" (qtd. in Haefner 328).

Haefner elaborates on this connection to Foucault's *Archaeology of Discourse* with an iteration of Shakespeare's "famous conditional statement" ("To be, or not to be?") written in the C programming language (328). His comparison is, as he concedes, "contrived," but he continues to analyze the binary, using it to contend that code is the construction of nothing but literal imperative statements—extreme binaries—with not nearly the personal and cultural fluxus, state of limbo, and/or interplay as the original Shakespearian version. In the end, Haefner privileges Shakespeare's version over the imperative performance of the code, stating that "[a]s the code performs in the computer, bursts of electricity trip circuits; as Hamlet speaks on the stage, words flow, complicate and negate themselves, but no action takes place. Structured programming always chooses ‘to be’; Hamlet dangles between the two” (329).

Haefner’s analysis is incorrect and is also unfortunately indicative of the type of analysis that computer scientists might find all too humorous.[1] Haefner is correct to indicate that his example fails to capture the “dangling between the two” scope of Shakespeare’s construction of Hamlet in this moment, but I argue that such a “dangling” experience can be expressed procedurally in code, emulating such processes written into the “code” of the performance of Hamlet. Furthermore, his own procedure could become much more complex with additional source work to make it a much more dialogic and interactive environment. By writing a program that manages input/output from the users, orchestrating more complex procedures, recursion and networks, users could experience a re-enactment of Hamlet’s disposition of uncertainty, rather than simply remediating Shakespeare’s original soliloquy verbatim as a text.

Literacy scholar Annette Vee also makes this claim to code’s dialogic nature and closer relations to oral communication. She argues that “Code can result in more dynamic and interactive compositions than traditional text—for instance, games and interactive fiction—because of this ability to change paths based on input" (22). Conversely, Haefner’s analysis of code keeps code within a dichotomy, not only separate from the discourse of language as experienced out of code, but also out of its evolving role(s) in the age of the Internet, because, as this thesis will continue to argue, *whether one is in or out of the code*, these networked streams of processes create a much more dynamic, social

[1] As I have learned in regular conversations with those in the field of computer science.

environment than Haefner's constrained analysis with an example that partitions the individual off from the social experience of the networked environment. Unfortunately, Haefner's critique of code still remains emblematic of both the field and public's perception of code as a tamable, less complex thing to be learned and written. Many writing teachers understand that teaching grammar does not produce good writing, and Kay has been arguing since his the 1970s that the same principle can be applied to programming.

More recently, Bradley Dilger, in "Beyond Star Flashes," discusses how writers (of traditional texts) in a networked culture, simply by becoming more critical of the spaces they use, "can engage the same manipulations and produce the same results as those who are fluent users of code—and if they choose, leverage these affordances into learning" (21). He argues that there is a reciprocal relationship between how "[c]ode begets function and vice versa," where both influence each other (21). He contends that many people choose to ignore how to learn code, creating the conditions necessary for technical communication, which "often involves writers doing the work of coding on behalf of readers, thereby facilitating the establishment of relationships between writers through *intermediary readers*" (emphasis added, 21). At this point, our field comes full circle back to the proposed trajectories by influential scholars in the early 1990s (Selfe and Selfe; Hawisher), where we remain writers *writing on the surface of the interfaces* that we may aptly *read* as problematic—even colonizing. Rhetoric and Composition is still restricted in our computer-mediated scholarship and praxis, because we failed to develop a more comprehensive outlook on the new inscriptive capabilities of the networked computer. In the end, we are restricted to one form of writing, caught within the linguistic discourses, since we overlooked the call to recognize, as McLuhan and Kay did, the new potential for procedural-based forms of persuasion and expression.

Conclusion: Positioning Proceduracy

In the early 1960s, Perlis envisioned “procedural literacy” to teach computation as it could have been enacted across the university just as rhetoric and composition emerged as a field with ties to both computer literacy and writing across the disciplines. Since the first generation of Computers and Writing, Rhetoric and Composition has turned away from the computer’s processing power, writing upon the surfaces of proprietary programs, while Computer Science has neglected to address the technocultural implications of writing code. Now, in 2012, Rhetoric and Composition, and particularly the subfield of Computers and Writing, sees an opportunity to re-engage with Computer Science, to take up coding not simply to code, but to extend the rhetorical power of our students and ourselves as a field via “proceduracy.” This concept, re-introduced in a significant way by literacy scholar Vee (2010) will be explored and explained at length in the next chapter.

CHAPTER 3: PROCEDURACY ACROSS THE DISCIPLINES

Writing Connections with the Metamedium

In 1960, in the introduction to *Explorations in Communication: An Anthology*, McLuhan wrote, “Today we’re beginning to realize that the new media aren’t just mechanical gimmicks for creating worlds of illusion, but new languages with new and unique powers of expression” (2). A new medium gives rise to new capabilities for modes of learning and the construction of persuasive and expressive elements. While the Code Year movement in 2012 attempts to motivate people to learn how to code, it’s educational mission is still informed by the old rhetoric of crisis. More over, the parallel histories of Computer Science and Rhetoric and Composition indicate that a wider perspective on the computation and the computer as a metamedium has yet to be fully explored by rhetoricians and compositionists.

Accordingly, in this chapter, I submit that new work toward a procedurally informed curriculum is made in Rhetoric and Composition. Starting with Vee’s work in proceduracy as framework, I will explain how proceduracy differs, but is connected to traditional modes of literacy. Finally, I will provide some examples that fall across Vee’s proposed levels of proceduracy (low, medium, and high) to hint at the possibilities for a curriculum that moves toward the making and theorizing of persuasive and expressive elements in code.

The Continuum of Proceduracy: Writing Procedures

To arrive at a clearer definition of proceduracy, I will begin with the recent call by contemporary scholars. Vee extends Perlis’ call for “procedural literacy,” re-naming it as *proceduracy*[2], because "Like literacy, proceduracy is a human facility with an expressive technology that is affected by intersecting social, cognitive and technological factors and can be used for creative and rhetorical purposes" (14-15). According to Vee, for people to become procedurally literate, they draw from their own exigencies, envisioning a process that they "want the computer to enact, then [they] break that process down into hyper-explicit procedures and encode them in a grammar and syntax the computer can understand" (16). She connects proceduracy with some of Perlis' original ideas on how it works across the curriculum, drawing intersections between it and multimodal writing, stating "Procedural compositions are often a combination of code and text, along with audio, visual and kinetic expression"—a constitution of the senses (80). Considering how code is the driving force behind these semiotics[3], Vee submits that the "symbolic task" of proceduracy takes on some of the following goals that are different from, but connected to, the epistemic work in traditional literacies:

> … procedural expression can be a more efficient way of encoding and decoding complex processes or handling vast quantities of information in non-linear ways, and can allow people to offload some of the information-processing onto machines. In the hands of a highly procedurally literate person, the computer can create worlds, support interaction, and manage

[2] From this point forward, I will refer to procedural literacy as proceduracy, but note that the two are the same idea.
[3] Semiotics, of course, refers to the study of signs and their many relationships with the social and cognitive structures, but there is also a branch of semiotics, Computational Semitoics, which typically studies artificial intelligence and knowledge representation within the subfield of Human-Computer Interaction. (See Tanaka-Ishii's *Semiotics of Programming* for more details.)

information on a scale far greater than linear composition tools such as text. (4)

To ground these claims for code's ability to "scale" all of these literate, discursive practices through the practice of writing code, she draws from Heidegger's *Being in Time*, contending that "computers are not objectively removed from human context; especially since they are constituted in human language [via the thousands of different programming languages], they are fully embedded in the world" (20). In code's simplest sense, programmers know code as "source" or "binary"—binary being the core of any program software that is then produced by compiling the source code. Vee notes how computer science often portrays computers as objective information-process machines. Yet, she highlights how "the subjective cannot be fully separated from the objective," since code derives from human language (20).

Vee makes some important distinctions between speech, writing, and code, claiming that, "Like writing over speech, there are tradeoffs to programming over writing as a method of expression: richer context and nuance is sacrificed for scalability, longevity and efficiency of dissemination" (11). She draws connections between writing and programming, indicating that both are technologies "that allow thoughts to travel beyond their thinker"—think of it as the scaling of our ideas (11). Logically, then, code also circulates and performs in the world on a far greater scale than either speech or writing has done before. While the scholarship in Rhetoric and Composition has made strides in theorizing the effects of the writer writing in digital, networked spaces, it, as a field has yet to explicitly explore the actual writing and contextualizing of computer code.

To open up the field to the complexities of code, tearing down the prevalent dichotomy typified by Haefner's conception of code, Vee elaborates on this new form of procedural expression, saying that proceduracy "emphasizes the knowledge of procedures (action) over description [(text)] in code…" (20). In *Persuasive Games*, videogame researcher, designer and critic (with a Ph.D. in comparative literature) Ian Bogost argues that computers, as a "flexible inscription medium," enable us to write and "define the way things work: the methods, techniques, and logics that drive the operation of systems, from mechanical systems like engines to organizational systems like high schools to conceptual systems like religious faith" which simulate such systems and highlight Kay's recognition that the computer is, indeed, the first metamedium (Kindle Locations 176-178).

Recalling the history of Rhetoric and Composition, even Computers and Writing/Composition, we have yet to explore the writing of the rhetorical and expressive power of processes and computation and its connection to the multimodal forms of writing we theorize and practice on the surface-level screens. Our history indicates that we have yet to play a role in the shaping of the procedurist, who has an evolving set of skills that explore, as Vee writes, "the range of process-knowledge a person can draw on in the particular domain of computers" (46). Through the scope of programming languages, Vee posits a "multivalent continuum" (47) of proceduracy that explores how, "[a]t each 'level,' a person's knowledge of how data and processes can interact increases" (46).

Before she proposes such a continuum, she recognizes that, like any other form of literacy, proceduracy is subject to "issues of measurement," standardization, and what it means to be procedurally literate (45). She acknowledges Harvey Graff's arguments in *Labyrinths of Literacy* on perceiving "literacy in terms of its spread amongst the masses of people and what the ideological impetus behind the spread was, rather than on levels of literacy in particular people, or how useful those skills were for individuals" (2010, 45). Yet, she "refus[es] to be backed into a corner" by calling upon literacy historian David Vincent's contentions in *The Rise of Mass Literacy* that to be literate "is a different experience than [being illiterate]" (46). Vee, like Vincent and myself, "find the value in trudging on regardless" of Graff's justifiable concerns (46). Considering these concerns, I will now provide an overview of Vee's continuum of proceduracy to then use it as a method to locate the position, (i.e., level), of both Rhetoric and Composition and Computer Science on the continuum, so I can provide a clearer vision as to how and why we must begin to enact proceduracy across the disciplines in the university.

The Levels of Proceduracy

Responding to the two histories that provide a picture of the stalled revolutions around proceduracy in both Computer Science and Rhetoric and Composition, I will now build on the calls for a proceduracy-informed path that has been around since the 1960s, but has largely failed to scale in the educational system. This call to code across the disciplines is more than Code Year's call to code, because it will fold in the recognition of the rhetorical and expressive nature of code that, arguably, has been around since the conception of proceduracy. What I hope to provide in this chapter is a vision to do more of this kind of work in Rhetoric and Composition to increase our rhetorical faculties and incorporate a far more robust sense of multimodality than any type of Code Year movement or lone Computer Science model could provide on its own accord.

I readily accept the reality to the slow building of the infrastructure required of this vision of proceduracy across the disciplines in the university, but I will note that I cannot cover the more bureaucratic/political details due to the goals indicated in this thesis.

Instead, this chapter reviews some of the smaller installments of proceduracy that I argue must happen more often and more widely dispersed across the university to stimulate the conversation on learning how to code beyond the currently dominant paradigm shouldered by Computer Science and the movements, such as Code Year that respond too quickly to the looping discourse of crisis and ultimately re-enact the current programming paradigm in Computer Science. Now, instead of simply writing or remediating curricula, as Codecademy does with learning how to code, I will use Code Year as a *kairotic* moment to stimulate the greater potential behind proceduracy and the persuasive and expressive nature of code.

Vee begins the discursive process to define some metrics on measuring proceduracy "in terms of programming, which indexes a fluency with process-knowledge" with the following three main levels: 1) Low: Navigation and Customization, 2) Medium: Modification, and 3) High: Writing code that generates materials, tools, and/or simulations for users (47-48). Please note that I have ascribed the low, medium, and high levels in a more explicit way than Vee, as she described both low and high levels in a much more subtle way in her dissertation. Furthermore, I attached the term "medium" to the skill of modification, as listed above; more generally, Vee did not define modification as a medium level, as this is my language.

Low Levels of Proceduracy

In the low level of proceduracy, Vee argues that *navigation*, the ability to use menus and "successfully navigate the options in a program," is the lowest form of proceduracy (47). She notes how navigation is not traditionally included within the continuum of learning how to code, but contends that it belongs on the "proceduracy scale because it draws on an understanding of a task as broken down into explicit procedures, which can be standardized and provided as options on menus" (47). Next highest in the low-level range is *customization*, which is "the ability to change options on a computer to fit personal needs" (47). She writes that this skill "indicates a deep understanding of the interactive and malleable nature of the computer, and a knowledge about what nodes can be changed" (47).

I argue that these capabilities to navigate and even customize the interfaces and environments that we inhabit implies and/or equates with the reading and critical analysis of such coded spaces. As discussed in the history of Computers and Writing, many in this field are critically engaged in the spaces that we write (Dilger, Hawisher, Johnson-Eilola, Rice, Selber, Selfe, Sirc, and more), but not all have claimed that some of us should actually write the code that produces these spaces. Unfortunately, the argument to leave coding to other disciplines implicitly suggests that we, as a field dedicated to writing, remain *readers*. Computers and Writing, for the most part, has not even been reading the actual text that is code. Instead, most in our field are solely readers of the production of such processes undergirding the networked environments.

Yet, in the year of code of 2012, Rhetoric and Composition scholar James Brown, Jr. designed and taught a "Digital Rhetorics" course that took up Code Year's call and also extended it to incorporate Bogost's call to consider the persuasiveness of games, asking students to both read, analyze and eventually construct their own procedural arguments ("Political Procedures"). Although, before discussing Brown's curriculum in more detail, it is first important to define Bogost's "procedural rhetoric" to provide some context for Brown's call to code in a Rhetoric and Composition class.

In *Persuasive Games*, Bogost states that "[J]ust as verbal rhetoric is the practice of using oratory persuasively and visual rhetoric is the practice of using images persuasively… [procedural rhetoric is] the method of using processes persuasively" (Kindle Locations 707-708). He elaborates on this theory of rhetoric, saying that procedural "arguments are made not through the construction of words or images, but through the authorship of rules of behavior, [and] the construction of dynamic models (Kindle Location 711).

To make this case for procedural rhetoric, Bogost reviews new media, hypertext, and digital rhetoricians, such as Lev Manovich, Laura Gurak, Barbara Warnick, and Richard Lanham. Each of these theorists, as Bogost discusses, call for educators "to provide stylistic training in increasingly indispensable digital forms like email and the web" (Kindle Locations 650-651). He positions procedural rhetoric beyond the scope of digital rhetoric, arguing that, "procedural representation can muster moving images and sound, and software" (Kindle Locations 824-825). He develops procedural rhetoric further via video games because they "are capable of generating moving images in accordance with complex rules that simulate real or imagined physical and cultural processes" (Kindle Location 826). "Furthermore," writes Bogost, "procedural representations are often (but not always…) interactive; they rely on user interaction as a mediator, something static and moving images cannot claim to do" (Kindle Location 827).

In Brown's class, his students both rhetorically analyzed and produced videogames with the main goal to produce an "effective procedural argument" ("Group Project: Videogame"). On his syllabus and introduction to the course, he notes that "No specific technical expertise is required for this course," indicating that a person on any level of Vee's continuum could benefit and produce content for the course.

Recalling Vee's continuum of proceduracy, this type of class is able to accommodate the low to medium levels of proceduracy, asking students to read and navigate the procedural arguments of video games, then collaboratively design and build a game with procedural arguments. Reminiscent of Selfe and Selfe's "Politics of the Interface," Brown's class conducted a close reading of the video game called Braid, writing a rhetorical analysis paper in response to enacting Bogost's challenge to play/read games critically. Arguably, assignments to critically read, analyze, and write about the digital or new media spaces are relatively commonplace in Rhetoric and Composition, but Brown also asked the students to build their own games. Based on his primary goal to take up Bogost's challenge, he has deepened the scope of Code Year by both critiquing *and* writing procedural arguments being made about the culture in which our students are situated.

Upon reflection, Codecademy's "Code Year" movement is a reaction to the teaching methods of Computer Science and the university writ large on how to code. Yet, their initiative simply takes up coding in its more baseline, grammar/syntax sense, which Vee locates on the medium range on the continuum of proceduracy. Yet, I argue that the ability to critique and respond to the procedural connections between the computational platform and society, beyond just the digital rhetoric expressed epistemically on the surface of the medium, reveals the gap in Codecademy's learning initiative, but also the university's own system to provide more diverse ways to gain new levels of insight into what Bogost calls the "assemblages of procedural forms" (Kindle Locations 398-399). I submit that we can add new levels of sophistication to Vee's "multivalent continuum" by addressing the technocultural, rhetorical, and expressive qualities associated to the writing of code. In this moment, scholars who are interested in taking up this call in both Computer Science and Rhetoric and Composition can use this moment to help develop, to draw from Aristotle, new *any means available for persuasion* and expression.

At the low levels of proceduracy, it is vital for procedurists to develop an understanding of how the computer is much more than its pre-packaged or proprietary software. Instead, with a focus of it as a metamedium, new exigencies will be found to motivate learners to traverse beyond the provided spaces of software and begin exploring and modifying such "soft" spaces. While the subfield of Computers and Writing/Composition have been making strides in the lower levels of proceduracy, reading the politics of the interfaces and exploring the persuasive and expressive capabilities writing in the spaces of proprietary software, now its important for us to begin critiquing *and* writing our own procedural forms, which begins in the medium levels of proceduracy.

Medium Levels of Proceduracy

Vee argues that *modification* marks the moment, where users begin to conduct the more "traditional… work with code" (47). As such, the user modifies "small bits of code, even if only changing variables or simple properties," which Vee suggests "indicates a deeper understanding of how data and processes interact in the code" (47). Yet, what can now be added to Vee's levels at the medium level of proceduracy is the writing of *procedural arguments*, thanks in part to the variety of programming languages available to begin traversing this continuum of proceduracy.

During the four-week duration of the game-building phase of Brown's class, students used the programming language and environment called *Scratch*. Scratch is a higher level programming language that is also a visual, tile-based language that was designed to enable beginning programmers (notably children) the ability to construct the processes of programming without having to worry about learning the syntax/grammar of code.[4] According to Brown's class schedule, he provided students a Scratch workshop in every weekly class period throughout the series of three phases for each group's development of their game.

One of the completed group projects, available on Brown's website, is called "Walker, Wisconsin Ranger," which responds to Wisconsin's controversial 2011 Budget Repair Bill enacted by Wisconsin governor Scott Walker. The students stage the social context for the game, writing "The Bill had severe consequences for the collective bargaining rights of state workers," and as the students also indicate, Walker "has cut millions from government spending, including education, healthcare, government jobs, local government aid, Planned Parenthood and more" ("Walker, Wisconsin Ranger").

The game's procedural arguments attempt to illustrate the difficult nature behind keeping Walker's constituents at no less than a 25% approval rating, while balancing a budget, which in this case, is over three billion dollars in debt. To balance the budget, of course, requires making cuts to it, and the students reveal how the game represents the act of making budget cuts in the following three ways:

> First, it tracks the amount of money left needed to balance the budget. Second, Walker's approval rating fluctuates in proportion to his cuts. Third,

[4] Scratch also derives from Alan Kay and Alene Goldberg's original object-oriented programming language, Smalltalk, which also finds its roots in Seymour Papert's Logo, which both became the initial programming languages to be introduced to children for educational purposes. (For more information, see Kay and Goldberg's "Dynamic Personal Media" 1977.)

the cuts affect the population of badgers protesting Walker in the final scene of the game. ("Walker, Wisconsin Ranger")

When playing the game, I could see these particular arguments as I chopped away at the budget. To some extent, I immediately understood what cuts I could make, since the trees each bear a symbol on it to indicate the type of cut Walker can make, whether medical, educational, state/city government, or workers union. After making a cut, two word bubbles would appear to indicate both the program and amount cut from the program, which I could then see how much the constituent approval went down, as seen in Figure 1.

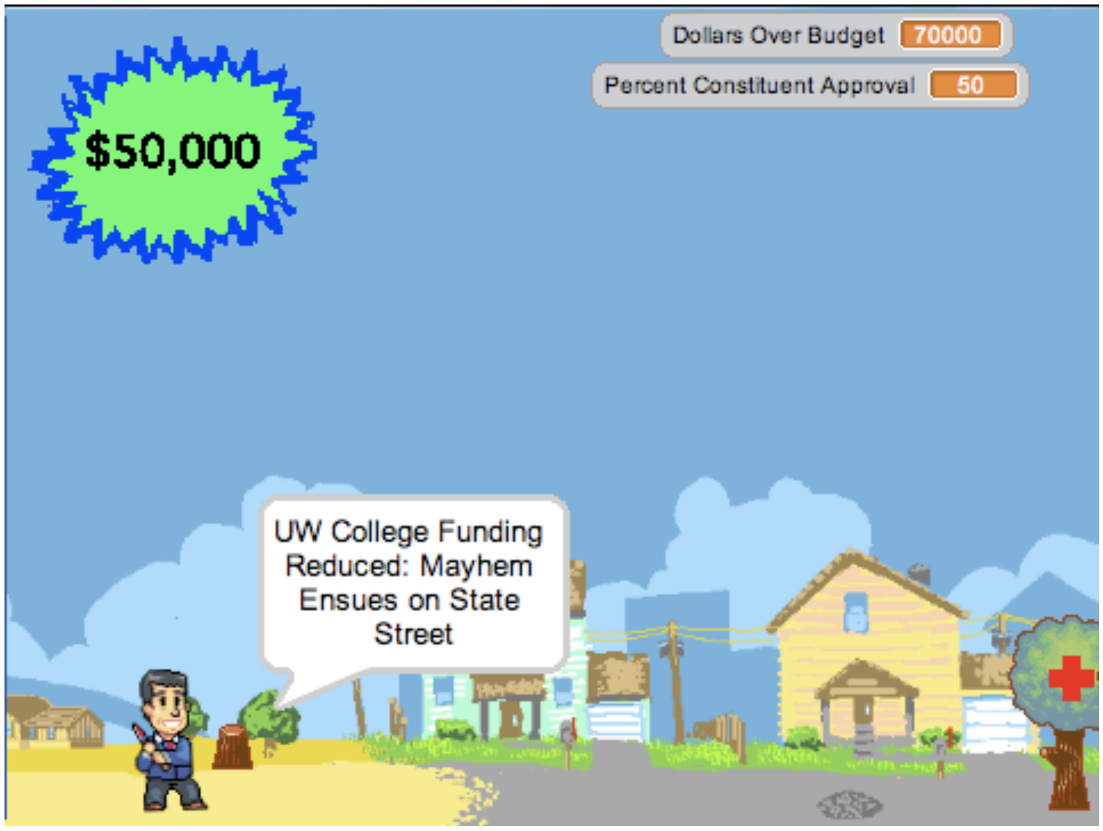
Dollars Over Budget 70000
Percent Constituent Approval 50
$50,000
UW College Funding Reduced: Mayhem Ensues on State Street

Fig 1. Screen capture of "Walker, Wisconsin Ranger." *Jim Brown's Courses*, 2012.

I thought one of the more interesting arguments made in this game was rooted in what was left out of the more explicit instructions and even the description of the game. After playing the game a few times, I realized that it wasn't just the dollar amounts that mattered, but moreover the particular nature of the government program. For example, it is far more advantageous to cut the city government's sanitary management services, which equates to a $20,000 budget cut with a decrease in 5 constituent points, versus taking union workers' rights away at minus $10,000 with a decrease in 15 constituent points. As I played the game further I was then able to recall which programs embodied what set of values held by the constituents, as indicated by the ratio between the budget cut to its point decrease. To win the game, you must figure out what are the most advantageous ratios that represent the values of Walker's constituents. Essentially, the students developed a series of procedural arguments that represented how a politician must successfully appeal to the people who have a direct affect on their career: the constituents.

Bogost refers to this moment in critical gameplay as the "simulation gap" (*Persuasive Games*, Kindle Location 4300). He writes "The player's evaluation of [the] claims as depicted in the game's rules opens a simulation gap, a space of crisis in which the persuasion game plays out" (Kindle Locations 4300-4301). Now, I do not know what the prior programming skills were of this particular group of students, nor how the group dynamic functioned, but this game embodied the rules that the developers of this game think politicians abide by, i.e., politicians stay in office by pleasing their constituents. In this particular game, the procedures driving the experience of the game effectively argues the constraints of the situation by which these students see Walker bound by. In so doing, these students began to take some strides toward how Bogost constructs procedural rhetoric not simply as an extension of digital rhetoric, but as the underlying force driving such epistemic work with text and images, which is the primary concern for Rhetoric and Composition.

Furthermore, Bogost argues that the scope of digital rhetoric usually focuses on "the presentation of traditional materials—especially text and images—without accounting for the computational underpinnings of that presentation" (Kindle Locations 696-697). He reviews various digital rhetoricians, and contends that their strategies ultimately leave the properties of computation "black-boxed" and procedural rhetoric "is a digital rhetoric that addresses the unique properties of computation … to found a new rhetorical practice"—a practice that Brown has begun to take up, yet still needs to be fully explored in Computers and Writing/Composition (Kindle Locations 654-655). These students not only understood the political situation, but also understood it well enough to create an environment for other people to experience their position on the subject.

High Levels of Proceduracy

At the high level of proceduracy, a level that I would characterize as becoming a type of *procedurist*, Vee argues that, while she considers it the "final" part on her levels of proceduracy, it is not meant to place a "finite limit on [it]," rather, as she suggests, "proceduracy becomes impossible to pin down after [this stage]" (48). This stage includes "a more sophisticated awareness of expression of processes through the computer," as well as the implication for "a greater rhetorical sophistication"—a goal deeply connected to the field of Rhetoric and Composition, yet left relatively unexamined thus far. Although, as indicated in the prior medium level, procedural arguments can be written in the earlier levels of proceduracy as well, so how can the even "greater rhetorical sophistication" be characterized at higher levels of proceduracy?

Vee defines this higher level of proceduracy with the ability to write "behaviors that can be coded through recursion, or by nesting functions, or by creating classes," all of which, she suggests, “become complex and impossible to enumerate” (48). Vee emphasizes that procedurcy is not only the skills learned to simply write software, but also how it serves a way of understanding the pliability of the general computer. She expands on this function of proceduracy, saying that “Because we are forced to make the process or how-to knowledge highly explicit through code in order to communicate with the computer, this knowledge, which was present but tacit in many human activities prior to the computer, is laid bare at levels of high proceduracy" (49-50). In the following excerpt, she draws from Kay’s widely cited “User Interface” to demarcate the difference between reading and writing in a medium, both print and computer:

> The ability to ‘read’ a medium means you can *access* materials and tools for others. The ability to ‘write’ in a medium means you can generate materials and tools for others. You must have both to be literate. In print writing, the tools you generate are rhetorical; they demonstrate and convince. In computer writing, the tools you generate are process; they simulate and decide. (Kay qtd. in Vee 48).

Kay likens proceduracy to the construction of environments, and Vee’s high levels of proceduracy, regardless of the difficulty and issues derived from its measurement, signal a moment to reframe the writing of code as more than the ability to write software that compiles and is used to achieve a low level function, e.g., office suites that eliminate the drudgeries of writing and information management. It becomes more than writing source code and even more than Perlis’ vision for thinking in computational models, which has the goal to think beyond programming languages.

At higher levels of proceduracy, I argue that the performativity and circulatory skills of software and networks become particularly powerful forms of persuasion and expressiveness. Recall: Vee argues that the scalability of code as a type of writing has capabilities beyond the scope of speech and print in a networked culture. McLuhan's claim that the medium is the massage begins to take shape as writing environments via the writing of code in the metamedium of the computer shows us that we, as users, must *perform* certain operations that are coded into the computer environments that we inhabit.

Drawing from Gender and Identity scholar Judith Butler (*Excitable Speech* 1997), scholar Adrian MacKenzie, who studies the cultural trajectories of code, argues that programs, written in code, can become "code-objects" that effectively drive culture to perform certain ways, because it repeats and circulates the "authorizing context" (Butler qtd. in MacKenzie 81-82). This "authorizing context," as Butler contends, constitutes a "prior and authoritative set of practices" (qtd. in MacKenzie 82). As MacKenzie indicates, "The agential effect of performativity arises first of all… through the repetition and citation," which computers are the perfect medium, since the can repeat such performatives at a greater scale than speech or writing (81). MacKenzie adds that "Performatives also 'succeed' not only by citing, or enacting through describing, but by 'covering over'" the "'authoritative set of practices' which lend force to the enacting" (82). Code is persuasive, not only in its ability to cite, repeat and circulate its performative processes at intense speeds, but, perhaps, is "successful" due to its ability to efface its processes from the WYSIWIG surfaces—similar to the ways performativity becomes effaced within cultures outside and apart from code. As code multifariously and repeatedly cites/is cited, enacts/is enacted, circulates/is circulated within a networked, situated culture, it simultaneously produces and, as Butler argues of culture prior to the electric age, "echoes prior actions, and accumulates the force of authority" (qtd. in MacKenzie 77).

Here, I submit that procedurists, at this higher level of understanding of what code can accomplish rhetorically, can do far more than write standardized letters to the editor, or circulate petitions across the web with ready-made and/or pre-existing platforms to promote awareness or instigate a change politically. Now, beyond these means of persuasion and modes of discourses, they can pool together the people, data, and resources to draw from or write code libraries to create interactive applications that respond to rhetorical situations. In so doing, they can create multiple channels and access points for people to become involved and/or organize both on/offline to spark change in the sociopolitical sphere. One such group that I submit embodies these procedurate qualities is Queer Technologies.

Queer Technologies, established by media scholar and artist Zach Blas, is "an organization that produces products and situations for queer technological agency, interventions, and social formation" (Queer Technologies, "About"). I argue that this group embodies a core set of higher-level procedurist skills that have yet to be realized on the continuum of proceduracy. To fill the critical gap in the existing model where Computer Science shoulders the burden to learn how to code, I suggest that we draw from their goals and capabilities in and out of code, which also aligns with the socio-cultural work in Rhetoric and Composition.

This organization shows us some of the complexities of code and its relationship with culture that, as I submit, can be critiqued and analyzed in some fashion at any of the levels on the continuum of proceduracy, but also written in the code itself. In *Programmed Visions*, Software Studies scholar Wendy Chun discusses the emergence of programming languages and their performative role in software and cultural production, which characterizes the scope of Queer Technologies' goals. She reveals how the claim that "code is law—something [that] legal scholar Lawrence Lessig emphasizes—is hardly profound" (27). Instead, she argues that software code is "executable," and begins to illustrate how, at any given moment, no one truly understands how a computer performs its operations—even the most skilled programmers (27). She adds that "this executability makes code not law, but rather every lawyer's dream of what law should be: automatically enabling and disabling certain actions, functioning at the level of everyday practice" (27). Such power, originally designated to "the provenance of government," is enacted via code as well, "embod[ying] the power of the executive," which now *performs* both in and out of code (27).

Chun explains that complex code "is both specific and nebulous, both defined and undefinable," re-presenting programmers (and users) across the continuum of proceduracy "the fact that we cannot know software" in the singular sense that was once coded into the disciplines, as remains evident in the two forestalled revolutions. Instead, Chun calls for everyone to embrace this paradoxical nature of code, as "an enabling condition: a way for us to engage the surprises generated by programmability that, try as it might, cannot entirely prepare us for the future" (54). While Chun reveals how programming languages are socially-situated constructions that also give rise to unexpected outcomes, Vee articulates "that code comprises virtually all of our current and "new media" compositional environments, and that the literacy connected with this writing—proceduracy—embodies a new, very powerful and highly rewarded means of expression, makes it critical for us to study" (5).

Drawing from Chun, Queer Technologies connect code and the ideologies of culture within the scope of Chun's advice to not tame the procedural claims of code, but, instead, learn from them and enable such theories, ideologies, and/or subjectivities to be more fully conceptualized and experienced through this reading and writing process. Blas writes about some of these goals in his M.F.A. thesis for his "transCoder" project:

> Specifically, as a queer software application, transCoder is devoted to rupturing the heteronormative superstructure that has infiltrated coding and software historically, discursively, and culturally. transCoder strives for a complete shattering of code's ontology. Viewing transCoder as a "language" battle between seemingly disjunctive fields of discourse (computing and queer theory), the application wants to sever ontological and epistemological ties to dominant technologies, to interrupt a flow of circulation between heteronormative culture, coding, and visual interface. transCoder stretches out to the sublime of destruction—a desired ontological rupture of functionality, designed to initiate a conceptual reassessment beyond the technical.

Essentially, I argue that Queer Technologies is one example of a group of highly procedurate people, who have a nuanced rhetorical-scape, (i.e., both epistemological, utilizing traditional forms of writing, *and* ontological, writing code in response to other forms of procedural claims in the sociopolitical sphere), from which to review and draw from as a mode for new ideas for a proceduracy-informed educational system. Coupled with their call to code against the "dominant technologies," Queer Technologies also aids in their performative and circulatory power by supplementing their software and programming projects with more traditional print materials such as manifestos and websites explaining their actions and art installations in public spaces. Scholars, practitioners, and students in both Rhetoric and Composition and Computer Science can benefit from their procedural claims and discursive efforts to provide alternative and more robust enactments of the procedurate identity. They also provide their code libraries to the public for free, serving as a rich source for scholars and practitioners to examine and build on their procedural forms and claims.

Connecting even closer still to Rhetoric and Composition, the first wave of Computers and Writing wrote their own homegrown software, but, as discussed in chapter 2, proprietary software soon attenuated such efforts throughout the 1990s. Yet, some scholars and educators in the field are still pushing forward in this domain of higher levels of proceduracy, where writing teachers are writing software once again to help our students become even better writers in the traditional literacies of print culture.

Bill Hart-Davidson, Jeffrey Grabill and Mike McLeod at Michigan State University's WIDE (Writing in the Digital Environments) research center have recently developed and implemented a process-intensive, peer review software program, ELI, that enables students to improve writing and teachers improve methods of instruction. ELI incorporates the use of real-time, user-defined data and concurrent folksonomic analysis to provide teachers and students with the immediacy of data-driven moments of learning in peer review workshops. Drawing from Lev Vygotsky's theories of learning, ELI's software creates an environment where and when student and teachers can take advantage of the affordances of learning from "a more capable peer" as everyone is writing feedback on other people's writing (Hart-Davidson and Grabill, "Learning Theory and Writing"). ELI's real-time results, thanks in part to the process-intensive environment that organizes and displays such results for everyone to see, enables students to immediately diagnose their writing issues, (which are also pre-defined by the teacher-user), as they are in the act of writing. Hart-Davidson equates ELI's effectiveness to the comparison of watching a person learn how to dance with a group of people. If one person forgets or misses a few steps, they simply look over at the group to quickly fall back into the rhythm. The same approach can be applied to learning how to write, argues Hart-Davidson and Grabill, with the excellent, process-intensive designs.

I suggest that there are even more possibilities for ELI, such as having students analyze the performativity of ELI's procedural methods to improve their own writing and critiquing of other peoples' writing. Akin to Seymour Papert's constructionist principles of learning in *Mindstorms*, while avoiding the issues that Bogost argues surrounds Papert's philosophy's dismissal of the importance and benefits of social-contextual learning (Kindle Locations 4345-4439), proceduracy can help students learn how to write better by also critiquing the very technologies that attempt to make them better writers. By offering students the opportunity to read and develop better procedures, writing pseudocode at first, then with students who are higher on the continuum actually help to write and document the new code, better writers can emerge from the construction of their own processes of learning. With these examples in mind, the call for writing teachers to take up writing software again seems like the next logical step in creating the most effective environments to not only develop higher procedurate skills, but also teach and learn literate-writing skills as well.

.

CHAPTER 4. CONCLUSION

"The best way to predict the future is to invent it." (Alan Kay)

Now, arriving at the conclusion of this thesis, this chapter has two goals. The first is to summarize the some of the conclusions about proceduracy that are a result of this history and theory paper. The other goal is to cover some of the implications of proceduracy and Rhetoric and Composition's role in building its infrastructure, which includes my position that our field should consider becoming involved in the following three research and development areas: 1) theoretical connections to rhetoric, media and writing, 2) a proceduracy-informed curriculum, and 3) proceduracy prior to the university.

My main goal with this thesis was to not only show some of the early good work that has already been accomplished along the peripherals of the main pipeline of learning how to code by scholars in both Computer Science and Rhetoric and Composition, but also that some of us in Rhetoric and Composition have a role to play in development of a more nuanced sense of rhetoric and writing that folds in the *reading, rhetoric and writing* of code and culture.

I hope it conveys that the goals of proceduracy are not just to arbitrarily learn how to code in one year, nor, as the first generation of Computers and Writing/Composition attempted to do, write software that sought to eliminate the drudgeries of grading, data collection, assembly and analysis, as well as epistemic work towards the presentation of it. Proceduracy is also not simply the continuation of Rhetoric and Composition's critical readings of how our culture is networked, producing and circulating information in an information age. While I concede that this work should be pursued further by some of us in Rhetoric and Composition, I hope that it is now clear that the work of those of us, who are so inclined, must also include goals toward addressing the rhetorical faculties of the thousands of available programming languages and their various hardware counterparts.

Due to the scope of this thesis, I was only able to address a select few of the major issues surrounding the missed opportunities of proceduracy and the media turn as they happened in the 1960s and into the more recent possible continuation of this missed opportunity in 2012, "Code Year." Additionally, I was unable to address a central work that deals with processes, Noah Wardrip-Fruin's *Expressive Processing*, which underscores the essential role of interpreting processes in digital media and literature, but I chose to omit this work only because I desired to focus on rhetoric's role in the composition of process-intensive works. In chapter 3, I extended and supplied more shape to Vee's call to begin building the infrastructure of proceduracy by defining the procedurist at three different levels coupled with more recent theoretical developments surrounding the rhetorical nature of code and the metamedium of the computer.

Now, before discussing some of the implications of this paper, here are the major points that I trust this history and extension of proceduracy has made:

- *A clearer sense of the problem driving the proceduracy crisis*. Currently, I suggest that the Code Year movement marks a new moment in the continuum of literacy, when our culture is sensing another Trimburian "cultural lag," where coding is the new skill necessary to compete in the future local and global markets. Yet, the methods to learn these skills have yet to be truly updated and the discourse surrounding its importance keeps looping the same messages as the previous discourse of literacy crisis.

- *A better understanding of the history of programming and computers in the university*. Through the scope of two parallel histories in both Computer Science and Rhetoric and Composition, the 1960s proves to be a remarkable time period to reframe the so-called revolutionary uses of computers in educational contexts and its deeper connection to rhetoric and writing.
- *A more nuanced definition of proceduracy*. By providing some examples of proceduracy along its, as Vee calls them, "multivalent" levels, procedural skills now fold in rhetorical capabilities, where the rhetor writes procedural forms that create environments that express points of view and persuade its users to perform a certain role in a situation, exploring McLuhan's *massage of the medium*.

Using these major points as a guide, the remainder of this chapter will address the implications of this reframing of Code Year to incorporate proceduracy into its goals in and beyond the scope of 2012.

Theoretical Connections

At the 2012 Computers and Writing conference, Alex Reid's keynote address, "Composing Objects," pressed the field (Computers and Composition) to consider the potential for writing to live beyond linguistic discourse and, instead, begin to notice the power behind the "realization that rhetoric was never and could never have been a solely human province." He desires to push the nature of rhetoric and writing beyond simply being a hammer that sees a nail," where the composition of objects and "rhetorics … incorporate technoscientific objects and build a future that includes them [objects] rather than divides them from us." He emphasizes the hammer/nail reference as a means to draw attention to our previously held Modern-informed theories and practices as a "minimal rhetoric," which, as he submits to the field, is by no means a "wholesale rejection of the theories and philosophy that have shaped our field but rather a recognition of their limits in addressing rhetorical challenges that we can no longer afford to imagine as simply discursive." In conclusion, he positions the field to implement digital rhetoric as a method to investigate "the rhetorical operation of these objects so that," as he continues, "we might understand how our democratic, scientific, and cultural discourses develop with these objects as participants."

Here, I suggest that the digital rhetoric that Reid submits include the rhetoric of procedures, which if coded into the computer medium, can simulate our theories on the "rhetorical operation of these objects," as Reid states. The real exciting nature behind the pursuit of proceduracy in Rhetoric and Composition is the potential to explore Reid's claims, (which drew heavily from Bogost's work), in the classroom through the creation of persuasive simulations and environments. Accordingly, this call for proceduracy takes up a combination of the media turn that Rice reminds our field to return to, as well as Perlis and Kay's call for a more culturally sensitive and expressive outlook of the computer as the metamedium, not simply pre-packaged tools to consume.

Resources for a Proceduracy-Informed Curriculum

As reviewed in chapter 3, Brown extended the call to code in 2012 by also taking on Bogost's call to play and develop games more critically via procedural rhetoric. As to what more is to come, I seek to provide some more resources for those interested in developing their own curriculum. To cite Brown's efforts once more, he is currently involved in the development of a course akin to Perlis' original call for a class that focuses on computation for incoming first-year students to the university. In conjunction with the

Computer Science department, he is developing a series of courses for first-year students, “Writing and Coding: Composition, Computation, and New Media Studies” that will be interdisciplinary in scope, but will explore the relationships between writing, coding, and media.[5] In Spring 2012, Digital Media and Communications scholar Kevin Brock designed an upper-level humanities, special topics course, “Code, Computation, and Rhetoric,” on digital media and rhetoric for English and Communication undergraduates. This 16-week course, which can be found in its entirety on GitHub.com, sought to explore the rhetorical and persuasive capabilities of software code. On the expressive side of processes, Mateas developed a course in Fall 2005, “Computation as an Expressive Medium,” that sought to take a more critical, media-focused approach to teaching the widely learned Java programming language (also available online). Each of these courses, including Brown’s course, incorporate their readings discussed throughout the course to supplement more critical discussions on the nature of writing code, which, on a personal note, I desired while learning Java in an Intro to Computer Science course.

Furthermore, for more advanced courses, recall that Queer Technologies provides numerous code libraries and a software development kit (SDK) available for download on their “Projects” page. For those interested in exploring the processes behind the visual rhetoric on the surfaces of our screens, the programming language Processing (Processing.org) is based on Java and includes a good amount of code libraries for beginners and a lively community and good amount of literature (“Processing: Learning”).

Regarding those interested in immersing themselves in the larger conversation, Vee has recently published an annotated bibliography, “Computer Programming and Literacy,”

[5] The hyperlinks to each of these resources can be found in the Works Cited.

of works related to computation and proceduracy, which is available to download on Scribd.com.

Proceduracy Before the University

If a more robust and multiple sense of the procedurist identity will be achieved, beyond the professional programmer identity, the work to develop a proceduracy-informed curriculum must be done prior to the university in K-12 setting. Since 2010, I have had the opportunity to co-design and co-implement a proceduracy-informed after-school program for 4th and 5th graders at a local school in Fargo, ND with Rhetoric and Composition scholar Kevin Brooks. Over the course of these two years of experience, collaborating with developers, educators and administrators from various disciplines, we have collected data in the form of videos of the afterschool sessions, coupled with observational notes from the researchers. While I am unable to report the extent of our findings, as we will be writing about them in two future publications, our initial findings related to the students' persistence and collaboration in response to the challenges of programming indicate a reliance on trial-and-error learning, rather than procedural thinking, which we think suggests that a longer-term, interdisciplinary approach is necessary to build the continuum of proceduracy prior to the university. Additionally, we have also seen the validity behind the power of the media ecology, i.e., the problems open-source projects, such as our own, face against proprietary systems, such as Windows, Mac OS, and mobile devices. Simply put, kids love to create, but more importantly, perhaps, they love to share their creations with their friends. Consequently, if the file formats are not compatible with other systems, the majority of children will not care to learn about the codecs (coder-decoders) necessary to get their programs to run on other platforms.

Accordingly, we argue that the public work of writing and literacy scholars and educators must engage and operate in contexts of literacy acquisition prior to the university, bridging stronger connections to reduce digital divides and transform K-16 students from consumers to programmers. From there, this small-scale pedagogical project gestures toward larger and longer-term vision to build a "smart computing culture" via proceduracy, which opens up collaborations with computer science and education, calling for us to begin teaching proceduracy across the disciplines to foster interdisciplinary collaboration within the public sphere. Such a challenge, we argue, requires us to reframe "code year" with a longer-term vision of "code decade," which would also lend force to new emerging procedurist practices to inform and shape the enactment of proceduracy in the university in the future.

Furthermore, the involvement of the humanities is relatively thin in this latest push for computational thinking. Upon closer inspection of Google's "Exploring Computational Thinking," their own stake in this educational movement in K-12 educational contexts, a search for English and Language Arts lesson plans currently yields only four results that are designed to learn basic grammar/syntax and spelling in a rather formulaic manner—a remediation of the old print-based worksheets. Unfortunately, this is reminiscent of the writing software created by the first generation Computers and Writing scholars, which puts a stark perspective on the real "cultural lag" surrounding proceduracy; especially, as it includes or excludes the humanities.

The Evolution of Writing Technologies

To end on an anecdote, I find it particularly intriguing that history tends to efface the original invention, abstraction, and intention of the traditional modes and technology of writing that we employ today from the cultures that use such an ever-evolving technology. In “Origins and Forms of Writing,” composition scholars Denise Schmandt-Besserat and Michael Erard write about the earliest archaeological evidence of writing systems that can be traced back to Neolithic sites in Syria and Iraq in 7,500 B.C.E., where the original intention of writing was actually connected to the practice of accounting (9). Farmers began to quantify their livestock, crops, and other possessions through the process of taking tallies inscribed by lines on clay or other abstractions, such as the use of disks or other items to represent a certain number of sheep or measure of grain. Now, we use counting machines, computers with incredible processing power, to code and compile such quantified abstractions of reality to accomplish far more than the original intentions of computers to simply be counting machines that find the next number in *pi*, or complete the human genome sequence. While, of course, these are amazing accomplishments made possible by the computer, it is my hope that we use Code Year to recognize the fuller breadth of rhetorical and expressive possibilities by helping to curate the procedurist identity with programming languages just as writers have continued to push the boundaries of the rhetorical and expressive power of writing in human languages.

I hope that my history and extension of proceduracy, while far from exhaustive, performs as a springboard for other scholars interested in this line of scholarly work to join in the research and development of the infrastructure necessary to enact a more critical approach to the medium that is so intertwined in our daily lives—the computer—via proceduracy.

WORKS CITED

Bogost, Ian. *Persuasive Games: The Expressive Power of Videogames*. Cambridge, MA: MIT Press, 2007. Kindle Edition.

Blas, Zach. "About transCoder." *transCoder*, 2010. Web. 12 May 2012.

Blas, Zach, Micha Cardenas, and Julie Russo. "Products: transCoder." *Queer Technologies*. 2011. Web. 12 May 2012 <http://www.queertechnologies.info/products/transcoder/>.

Brock, Kevin. Code, Computation, and Rhetoric. Course home page. April 2012. Communication, Rhetoric, and Digital Media, North Carolina State University. 27 April 2012 <https://github.com/brocktopus/engcom395>.

Brown, James. Digital Rhetorics. Course home page. Spring 2012. English Department, U of Wisconsin-Madison. 4 April 2012 <http://courses.jamesjbrownjr.net/550_spring2012>.

___. "Walker, Wisconsin Ranger." *Jim Brown's Courses*, Spring 2012. Web.

___. Writing and Coding: Composition, Computation, and New Media Studies. Course home page. Fall 2012. English Department and Computer Science, U of Wisconsin-Madison. 4 April 2012 <http://courses.jamesjbrownjr.net/LS102_fall2012>.

Chun, Wendy. *Programmed Visions: Software and Memory*. Cambridge, Mass: MIT Press, 2011. Print.

"Codecademy." *Codecademy.com*. Codecademy, 2012. Web.

"Code Year." *Codeyear.com*. Codecademy, 2012. Web.

Crowley, Sharon, ed. *Composition in the University: Historical and Polemical Essays*, PA: University of Pittsburgh Press, 1998. Print.

Davidson, Cathy. “Why We Need a 4th R: Reading, wRiting, aRithmetic, algoRithms.” *Digital Media and Learning Central*, 25 Jan. 2012. Web. 15 May 2012.

Dilger, Bradley. “Beyond Star Flashes: The Elements of Web 2.0 Style.” *Computers and Composition*, 27 (2010) 15-26.

Dilger, Bradley and Jeff Rice, eds. *From A to <A>: Keywords of Markup*. Minneapolis, MN: Regents of the University of Minnesota, 2010. Print.

Dworak, Wendy. “10 New Year’s Resolutions for Budding Digital Humanists.” *HASTAC.com*. Humanities, Arts, Sciences, and Technology Advanced Collaboratory, 31 Dec. 2011. Web.

Feldman, Stuart. “A Conversation with Alan Kay.” *ACM Queue*, 1 Dec. 2004. Web. 1 May 2012.

Gasch, Scott. “Alan Kay.” *Interactive Learning with a Digital Library in Computer Science Project*. Virginia Tech/Norfolk University, 1996. Web. 2 May 2012.

Gerrard, Lisa. “The Evolution of the Computers and Writing Conference.” *Computers and Composition*, 12, (1995) 279-292.

Google, Inc. “Exploring Computational Thinking.” *Google.com*. 2012. Web. 3 Jun 2012 <http://www.google.com/edu/computational-thinking/lessons.html>.

Haas, Christina. Writing Technology: Studies on the Materiality of Literacy. Routledge, 1995. Print.

Haefner, Joel. “The Politics of Code.” *Computers and Composition* 16, 1999, 325–339.

Haraway, Donna. *When Species Meet*. Minneapolis: Minnesota University Press, 2007.

Hart-Davidson, William and Jeffrey Grabill. “Learning Theory and Writing.” *Elireview.com*, 2012. Web.

Hawisher, Gail, Cynthia Selfe, Ahmed Shafinaz, and Gorjana Kisa. "Globalism And Multimodality In A Digitized World: Computers And Composition Studies." *Pedagogy: Critical Approaches To Teaching Literature, Language, Composition, And Culture* 10.1 (2010): 55-68. MLA International Bibliography. Web. 20 Dec. 2011.

Hawk, Byron. *A Counter-History of Composition: Toward Methodologies of Complexity*. Pittsburgh, PA: University of Pittsburgh Press, 2007. Kindle Edition.

___. "Toward a Rhetoric of Network (Media) Culture: Notes on Polarities and Potentiality." *JAC* 24.4 (2004): 831-50.

Inman, James A. *Computers And Writing : The Cyborg Era*. Lawrence Erlbaum Associates, Inc., 2004. eBook Collection (EBSCOhost). Web. 20 Dec. 2011.

Johndan Johnson-Eilola. "Control and the Cyborg: Writing and Being Written in Hypertext." *Journal of Advanced Composition* 13.2 (Fall 1993): 381-400.

Kay, Alan and Adele Goldberg. *Personal Dynamic Media*. IEEE Computer, (10)3, Mar. 1977.

Kay, Alan. "User Interface: A Personal View." *The Art of Human-Computer Interface Design*. Laurel, Brenda, ed. Reading, MA: Addison-Wesley, 1990, 191.

___. "The Revolution Hasn't Happened Yet." Keynote address. *ACM SIGPLAN Conference on Object Oriented Programming Systems, Languages, and Applications*, 1997. Web.

___. "Programming and Scaling." *Hasso Plattner Institute*, 21 Jul. 2011. Web. 08 Aug. 2011.

___. "Computer Software." *Scientific American*, 251(3), 1984, 41-47.

Kessler, Sarah. “NYC Mayor Bloomberg Vows to Learn Code in 2012.” *Mashable Business*. Mashable.com, 05 Jan. 2012. Web.

Knuth, Donald E. *Literate Programming*. Center for the Study of Language and Information, 1992.

Mateas, Michael. Computation as an Expressive Medium. Course home page. Fall 2005. Literature, Communication, and Culture, Georgia Tech. 6 June 2012. <http://www.lcc.gatech.edu/~mateas/courses/LCC6310-Fall2005/Syllabus2005.html>.

___. "Procedural Literacy: Educating the New Media Practitioner." *On the Horizon*, 13(2), 2005, 101-111.

Miller, John A. "Promoting Computer Literacy through Python Programming." University of Michigan, 2004. Web. Python.org. 7 Sep 2007. <www.python.org/files/miller-dissertation.pdf>.

McLuhan, Marshall. *The Gutenberg Galaxy: The Making of Typographic Man*. University of Toronto Press, 1962. Print.

McLuhan, Marshall and Lewis Lapham. *Understanding Media: The Extensions of Man*. University of Toronto Press: 2nd Edition, 1994. Print.

Papert, Seymour. *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books, 1980. Print.

President’s Council of Advisors on Science and Technology. “Designing a Digital Future: Federally Funded Research and Development in Networking and Information Technology.” Executive Office of the President, December 2010. Online PDF.

Raes, Cody and Ben Fry. “Processing: Learning.” *Processing.org*, 2011. Web. 6 Jun 2012 <http://processing.org/learning/books/>.

Reid, Alex. “Composing Objects” Keynote Address. *Computers and Writing Conference*, University of North Carolina, 2012.

Reider, David. “Introduction.” Townhall II Speaker. *Computers and Writing Conference*. University of North Carolina, 2012.

Rice, Jeff. “The 1963 Composition Revolution Will Not be Televised, Computed, or Demonstrated by Any Other Means of Technology.” *Composition Studies*, 33.1, (Spring 2005): 55–73.

Ryan, B., and D. Cutler. "Dynabook Revisited With Alan Kay." *Byte.Com* 16.2 (1991): 203. *EBSCO MegaFILE*. Web. 18 June 2012.

Salter, Anastasia. “New Year’s Resolutions: Learning to Program.” *Chronicle of Higher Education: Prof Hacker*. Chronicle.com, 10 Jan. 2012. Web.

Sample, Mark. “5 BASIC Statements.” Townhall II Speaker. *Computers and Writing Conference*. University of North Carolina, 2012.

Schmandt-Besserat, Denise and Michael Erard. “Origins and Forms of Writing.” *Handbook of Research on Writing: History, Society, School, Individual, Text.* Charles Bazerman, ed. New York: Taylor and Francis, 2008. 7-27. Print.

Selber, Stuart A. “The Politics and Practice of Media Design.” *Foundations for Teaching Technical Communication: Theory, Practice, and Program Design*. Ed. Katherine Staples and Cezar Ornatowski. Greenwich: Ablex Publishing, 1997. 193–208.

Selfe, Cynthia and Dickie Selfe. “The Politics of the Interface: Power and Its Exercise in Electronic Contact Zones.” *College Composition and Communication*, 45(4), Dec. 1994, 480-504.

Stolley, Karl. “Source Literacy.” Townhall II Speaker. *Computers and Writing Conference*. University of North Carolina, 2012. Web. 4 May 2012 <https://gist.github.com/2491521>.

Talbert, Roger. “Programming for All?” *Chronicle of Higher Education: Casting Out Nines*. Chronicle.com, 14 Jan. 2012. Web.

Trimbur, John. “Taking the Social Turn: Teaching Writing Post-Process.” *CCC*, Feb. 1994

Vee, Annette. *Proceduracy: Computer Code Writing in the Continuum of Literacy*. University of Wisconsin-Madison, 2010. ScienceDirect. 06 Jun 2011.

___. “Computer Programming and Literacy: An Annotated Bibliography.” *Scribd.com*. Jun 2012. Web. 7 Jun 2012. <http://www.scribd.com/doc/96306140/Computer-Programming-and-Literacy-An-Annotated-Bibliography>.

___. “Coding Values.” Townhall II Speaker. *Computers and Writing Conference*. University of North Carolina, 2012.

Wysocki, Anne Frances, & Jasken, Julia I. “What Should be an Unforgettable Face.” *Computers and Composition*, 21(1), 2004, 29–48.

Wysocki, Anne Frances, Johndan Johnson-Eilola, Cynthia Selfe, and Geoffrey Sirc. *Writing New Media: Theory and Applications for Expanding the Teaching of Composition*. Logan, Utah: Utah University Press, 2004.