Computer programming & literacy: an annotated bibliography

With the recent uptick in the "everyone should code" movement, it seems that everyone's now talking about computer programming as a new form of literacy. The terms by which people refer to the concept vary, but the central idea is shared: computational literacy; computational thinking; procedural literacy; proceduracy; computer literacy; iteracy. I've been working in this area for a few years now from the perspective of literacy studies, and I thought it might be a good time to share an annotated list of resources that I've found helpful in thinking through computer programming as a literacy. Chris Lindgren assembled a bibliography before me, and there's a lot of overlap here. (http://www.clindgrencv.com/resources/proceduracy-bibliography/) I'm inclined to say that the overlap points toward a burgeoning canon, although that recognition comes with the requisite wincing about a lack of gender/race diversity here.

I've listed just online or print texts, and the list tends toward the academic and historical. My Diigo library, assembled over the last few years with the tag "proceduracy", is a better resource for public discussions about computer programming as a literacy: http://www.diigo.com/user/advee77/proceduracy

I decided to list these in rough order of importance, which is incredibly subjective. I've broken the central sources up into a few categories: Really Important Stuff; Blogs & Online Writings; Dissertations; Work in English Studies. This is not to claim that there aren't overlaps (e.g., something can be important *and* online!) but just to organize it a bit. After the central list of sources for programming and literacy, I've included a list of related work that people might want to read in computer history, pop books, code studies, and composition & rhetoric.

Of course, the whole list is partial and biased! I welcome additions and reactions in the comments or via other contact media— email: annettevee@gmail.com twitter: @anetv website: www.annettevee.com .

Really Important Stuff

Papert, Seymour: *Mindstorms* (1980).

> Papert was a student of Jean Piaget and the main designer of the Logo programming language. In this book, he proposes that computers can be "objects to think with," that is, they can scaffold complex thinking about processes. He is

focused on formal education and children, in particular. The book is a wonderful, inspiring read about how children might be able to play and create with computer programs of increasing complexity.

The actual implementation of Logo didn't work in the way that Papert had hoped, and in response to the critique of the ineffectiveness of Logo in schools, he followed up by drawing a distinction between injecting a design into school, and growing a culture around it. Early criticism of LOGO (e.g., Pea and Kurland, 1984) assumed that Logo had failed because they weren't seeing the big picture; it "was in the spirit of those who dismissed the Wright brothers' flight on the grounds that a hop of 22 feet was of no serious importance."

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

Papert, Seymour. "Educational Computing: How Are We Doing?" T H E Journal 24.11 (1997). tlhle journal. Web. 23 Apr 2010.

Wing, Jeannette: "Computational Thinking."

Former chair of computer science at Carnegie Mellon University and current administrator at NSF, Jeannette Wing argues forcefully for a focus on "computational thinking" across all disciplines in the university. This widely-cited thought-piece is short and provocative. Wing's phrase "computational thinking" has perhaps enjoyed the most widespread uptake of all of the recent terms about computer programming as a kind of literacy, as evidenced by the "computational thinking" programs at Carnegie Mellon University and Microsoft.

Wing, Jeannette. (2006). Computational thinking. Communications of the ACM, 49(3), 33–35.

Mateas, Michael. "Procedural Literacy."

This article is focused on new media practitioners, but Mateas gestures outward as well: "In fact, one can argue that procedural literacy is a fundamental competence for everyone, required full participation in contemporary society, that believing only programmers (people who make a living at it) should be procedurally literate is like believing only published authors need to learn how to read and write." Great, tightly argued and highly-quotable piece from a computer scientist and co-author of the *Façade* game.

Mateas, Michael. "Procedural Literacy: Educating the New Media Practitioner." On The Horizon. Special Issue. Future of Games, Simulations and Interactive Media in Learning Contexts 13 1 (2005): 1-15. Print.

Perlis, Alan. "The Computer and the University."

This is the early piece that Mateas analyzes in his "Procedural Literacy" essay. As far as I know, as the proceeds from an MIT conference in 1961, it's the earliest articulation of computer programming as parallel to literacy.

Perlis argues for a course that would introduce all undergrads to the computer "because of the universal relevance of the computer to our times" (188). In fact, it should be central to the university mission: "The first and most critical stage of the computer's role in the university [...] is to train entering students in the theory of computation through the development of the concepts of programming." He outlines some basic principles of programming that haven't changed since then, such as: "the definition of complex processes by rational construction from simpler processes already given" (191). The discussion afterward between him and Peter Elias is also fascinating (for instance, Elias claims that language development will peak and end in just ten years from then!).

Perlis, Alan. "The Computer and the University." Computers and the World of the Future. Ed. Greenberger, Martin. Cambridge, MA: MIT, 1964. Print.

diSessa, Andrea. *Changing Minds.*

A wonderful book about "computational literacy," or programming to learn other subjects, by a former student of Seymour Papert. He argues that programming is headed toward being a new mass literacy, and he'd like to promote that. His central argument is (italics in original):

> *Computers can be the foundation of a new and dramatically enhanced literacy, which will act in many ways like current literacy and will have penetration and depth of influence comparable to what we have already experienced in coming to a mass, text-based literacy.*

He proposes three "pillars" of literacy (social, cognitive and material), and focuses on the material affordances of programming. He claims that a "material intelligence" can become a literacy when it becomes infrastructural to everyday life, and the ease of use of inscription systems can be the determining factor in whether something becomes infrastructural (he gives the example of Liebniz's more intuitive notation for calculus vs. Newton's). The body chapters focus on teaching physics with a programming language developed by diSessa (Boxer), but the intro and much of the book are incredibly useful for thinking about programming as a kind of literacy.

diSessa, Andrea. Changing Minds: Computers, Learning and Literacy. Cambridge, MA: MIT Press, 2000. Print.

Bogost, Ian. Persuasive Games.

This whole book is a great read and essential for any humanist interested in code, or anyone interested in studying video games—but the chapter on "Procedural Literacy" is what earns it a place on this list. Bogost doesn't emphasize coding per se, but sees the procedural aspects of games as a way into a kind of procedural literacy that has a lot in common with what others on this list describe: "procedural literacy entails the ability to reconfigure concepts and rules to understand processes, not just on the computer, but in general."

Bogost addresses James Paul Gee's work on what games teach: understanding how to move in semiotic domains. Bogost argues games teach *particular* and

contextual domains and mentions Gee's admission that what games really teach (at least games right now) is how to play that game. Like Gee, Bogost is interested in the potential of games to teach: "where the game you learn to play has a greater and more meaningful coupling with real experience" (240). He proposes procedural literacy as an entry to this. Bogost's website offers a nice summary of the book http://www.bogost.com/books/persuasive_games.shtml .

Bogost, Ian. Persuasive Games: The Expressive Power of Videogames. Cambridge, MA: MIT, 2007. Print.

Kay, Alan. "User Interface" and "Personal Dynamic Media."

At Xerox PARC, Kay was the mind behind Smalltalk, which was an important and early object-oriented computer language (the first one to be called that because it's Kay's term). He was also the mind behind laptop design, which he describes in the article with Adele Goldberg, "Personal Dynamic Media." This article is essential reading for anyone interested in the history of personal computing, but his article on "User Interface" is more relevant to thinking about computer programming as a literacy. In it, he writes,

> "The ability to 'read' a medium means you can *access* materials and tools created by 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." (193)

Based on McLuhan's argument about media (which he admits is faulty), Kay claims, "if the personal computer is truly a new medium then the very use of it would actually change thought patterns of an entire civilization" (193).

Kay, Alan. "User Interface: A Personal View." *The Art of Human-Computer Interface Design*. Ed. Laurel, Brenda. Reading, MA: Addison-Wesley Publishing Company, Inc., 1990. 191-207. Print.

Kay, Alan, and Adele Goldberg. "Personal Dynamic Media." *The New Media Reader*. Mar 1977. Eds. Wardrip-Fruin, Noah and Nick Montfort. Cambridge, MA: MIT Press, 2003. 393-404. Print.

Knuth, Donald. Literate Programming.

The influential computer scientists (and developer of LaTex word processing program) Donald Knuth famously argued that computer science should be thought of as more of an art than a science. Here's the gist of his argument about "literate programming":

> Programming is best regarded as the process of creating works of *literature,* which are meant to be read. […] Literature of the program genre is performable by machines, but that is not its main purpose. The computer programs that are truly beautiful, useful and profitable must be readable by people. So we ought to address them to people, not to machines. All of the major problems associated with computer programming—issues of reliability, portability, learnability, maintainability, and efficiency—are

ameliorated when programs and their dialogs with users become more literate.

Knuth's ideas expressed here have been very influential, along with the magisterial The Art of Computer Programming http://en.wikipedia.org/wiki/The_Art_of_Computer_Programming)

Knuth, Donald. *Literate Programming*. CSLI Lecture Notes. United States: Center for the Study of Language and Information, 1992. Print.

Bush, Vannevar. "As We May Think."

This is probably the most important historical essay about computers. In it, the WWII scientist Bush imagines what computers will do to change our lives: "The world has arrived at an age of cheap complex devices of great reliability; and something is bound to come of it." He proposes the Memex, a conceptual ancestor to library databases and the Internet, and an early articulation of the computer as an extension of the human mind. For the scholar, the Memex is *"*an enlarged intimate supplement to his memory."

Bush, Vannevar. "As We May Think." *The Atlantic*. 1945. http://www.theatlantic.com/doc/194507/bush/

Licklider, J.C.R. "Man-Computer Symbiosis" & "The Computer as Communication Device"

Licklider doesn't exactly talk about literacy here. But he does talk about the ways that humans and computers can work together, rather than thinking of the computer as an eventual replacement for human intelligence: "men are noisy, narrowband devices, but their nervous systems have very many parallel and simultaneously active channels. Relative to men, computing machines are very fast and very accurate, but they are constrained to perform only one or a few elementary operations at a time."

His co-authored 1968 "Computer as a Communication Device" is also wonderful, as he basically describes the idea of the Internet. In that essay, he claims that when people do their informational work "at the console" and "through the network," telecommunication will be as natural an extension of individual work as face-to-face communication is now (40). Perhaps tongue-in-cheek, he claims that life will then be happier and there will be no more unemployment because "the entire population of the world [will be] caught up inan infinite crescendo of on-line interactive debugging."

Licklider, J.C.R. "Man-Computer Symbiosis." *IRE Transactions on Human Factors in Electronics* 1 1 (1960): 4-11. *ACM Digital Portal.* Web. 23 Apr 2010.

Licklider, J., & Taylor, R. (1968). The computer as a communication device. *Science and technology*, *76*(2), 2.

Resnick, Mitchel, et al. "Scratch: Programming for All."

Nice, short introduction to the philosophy and implementation of the Scratch programming language, which is the spiritual inheritor of Logo's legacy.

Resnick, Mitchel, et al. "Scratch: Programming for All." *Communications of the ACM* 52.11 (2009): 60-67. *ACM Digital Portal*. Web. 23 Apr 2010.

Nelson, Ted. Computer Lib/ Dream Machines.

What a weird and wonderful book! Ted Nelson is the progenitor of hyperlinks, which inspired Tim Berners-Lee's design of the Web. Nelson first self-published this crazy, illustrated, and half-handwritten book and then (ironically) Microsoft republished it in a revised form (which is the copy I own and quote from). It's really hard to get ahold of, but I recommend trying!

Nelson describes himself as a fan rather than an "insider" for computers, but because he recognizes the importance of them, he wants them to be more accessible—he wants them to be "liberated." He portrays "computer people" as knowledge-hoarders, eager to keep the powerful device to themselves. He's not speaking specifically about programming, although computer usage was closer to direct programming when he was talking about this in 1974. Here's a long quote that I love:

> Somehow the idea is abroad that computer activities are *uncreative*, as compared, say, with rotating clay against your fingers until it becomes a pot. This is categorically false. Computers involve imagination and creation at the highest level. Computers are an involvement you can really get into, regardless of your trip or karma. They are toys, they are tools, they are glorious abstractions. So if you like mental creation, toy trains, or abstractions, computers are for you. If you are interested in democracy and its future, you'd better understand computers. And if you are concerned about power and the way it is used, and aren't we all now, the same thing goes.

He argues that people should rise up and demand that the computers and computer people don't control them: "THIS BOOK IS FOR PERSONAL FREEDOM, AND AGAINST RESTRICTION AND COERCION...COMPUTER POWER TO THE PEOPLE. DOWN WITH CYBERCRUD!" Fun reading!

Nelson, Theodore. Computer Lib / Dream Machines. Redmond, WA: Microsoft Press, 1980.

Brooks, Frederick P. The Mythical Man-Month.

This doesn't address literacy per se, but is so central to any discussion about programming and writing that it's on this list. Brooks, the main architect in the IBM OS/360 project, writes about modularity and scheduling large software projects, which tend to be extremely complex and impossible to predict. This should be read in the context of computer histories that cover this "software crisis" of the 1960s (say, Campbell and Aspray, or Ensmenger, see below). Brooks considers "the craft of system programming" and explores: why is programming fun? 1) The joy of making things, 2) the joy of making things for other people, 3) fascination with

interlocking parts, 4) you can always be learning because there are many non-repeatable tasks in programming. Perhaps the most-often quoted part of this book (at least in the humanities) is this bit:

> there is a delight in working in such a tractable medium. The programmer, like the poet, works only slightly removed from pure thought-stuff...Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures (5).

But, Brooks writes, programming is different from poetry: "Yet the program construct, unlike the poet's words, is real in the sense that it moves and works, producing visible outputs separate from the construct itself" (7).

Brooks, Frederick P. *The Mythical Man-Month: Essays on Software Engineering*. Reading, MA: Addison-Wesley Publishing Co., 1982. Print.

Nardi, Bonnie. A Small Matter of Programming.

An early 1990s-visionary book. Nardi is interested in "end-user programming" (EUP) which is basically the ability of end-users to modify and control the software they use. Her research group's goals for EUP are lofty: for EUs to "perform their work more efficiently, effectively, and pleasurably" but also because they can (no guarantees) learn the "nature of the machine—its possibilities and limits" through EUP. (3) The beginning is the most useful for explorations of code and literacy. Some good quotes from it: "While computer agents will be appropriate and useful for many tasks, such agents do not begin to cover the extent of what end users will be able to do with their computers when they have suitable environments for creating their own applications." (3) And her political focus is interesting to me: EUP is important "so that the many decisions a democratic society faces about the use of computers, [4] including difficult issues of privacy, freedom of speech, and civil liberties, can be approached by ordinary citizens from a more knowledgeable standpoint." (3-4) There's a lot about user interfaces here though, and much less about politics.

Nardi, Bonnie. *A Small Matter of Programming: Perspectives on End User Computing*. Cambridge, MA: MIT, 1993. Print.

Rushkoff, Douglas. *Program or be Programmed*.

I'm going to be honest: I found this book disappointing. It's got a provocative title and an interesting introduction and conclusion, but everything in-between was thin. But I like the way he talks about shifts in paradigms for participation in democracy, and that there are things one can't even conceive of if one doesn't know how to program. In addition to the title, there are some good, quotable passages. To wit:

> In the emerging, highly programmed landscape ahead, you will either create the software or you will be the software. It's really that simple: Program or be programmed. Choose the former and you gain access to the control panel of civilization. Choose the latter and it could be the last real choice you get to make.

And:

> Just as words gave people the ability to pass on knowledge for what we now call civilization, networked activity could soon offer us access to shared thinking--an extension of consciousness still inconceivable to most of us today. The operating principles of commerce and culture--from supply and demand to command and control--could conceivably give way to an entirely more engaged, connected, and collaborative mode of participation.

If you're interested in the idea of programming and literacy, read it—but maybe on the stairmaster like I did. Or you can check out his short article on Huffington Post, which promotes the book and outlines its argument: http://www.huffingtonpost.com/douglas-rushkoff/programming-literacy_b_745126.html

Rushkoff, Douglas. Program or be Programmed. OR Books, 2010.

Prensky, Marc. "Programming is the New Literacy."

Marc Prensky is the pop-educational theorist behind the "digital natives" term that most technology educators now groan about. As with his other work, this is pretty shallow in terms of analysis and historical reference, but it's darn quotable:

> As programming becomes more important, it will leave the back room and become a key skill and attribute of our top intellectual and social classes, just as reading and writing did in the past. Remember, only a few centuries ago, reading and writing were confined to a small specialist class whose members we called scribes.

This is a readable general overview of the topic without the political urgency of Rushkoff. And Prensky was a little ahead of the curve in pop-talk about coding as literacy.

Prensky, Marc. "Programming is the New Literacy." *Edutopia*. 13 Jan 2008. http://www.edutopia.org/literacy-computer-programming

Blogs & Online Writings

Guzdial, Mark. Computing Education Blog.

Georgia Tech computer science educator Mark Guzdial has a sustained interest in teaching computer programming to non-CS people, and blogs about it occasionally on this site. Lots of good stuff about procedural literacy, which is his preferred term.

Guzdial, Mark. *Computing Education Blog*. Wordpress. Web. <http://computinged.wordpress.com/>.

Why the Lucky Stiff. "The Little Coder's Predicament."

Why the Lucky Stiff was the pseudonym of a prominent member of the Ruby programming language community and author of a crazy, Ted Nelson-inspired book on Ruby, "Why's Poignant Guide to Ruby," who disappeared from the scene in 2010 (he's all right and if you're curious, you can read more about him in this

Slate article
http://www.slate.com/articles/technology/technology/2012/03/ruby_ruby_on_rails_and__why_the_disappearance_of_one_of_the_world_s_most_beloved_computer_programmers_.html).
This little gem of a piece talks about what it's like to learn programming now, versus the BASIC days. Why has some other wonderful pieces online that you can access here: http://viewsourcecode.org/why/ as well as a wonderful video from the 2009 Art && Code Symposium, where I was lucky enough to encounter him: https://vimeo.com/5047563

Why the Lucky Stiff. "The Little Coder's Predicament." 11 Jun 2003. *Avogato*. Web. 23 Apr 2010. <http://www.advogato.org/article/671.html>.

Perlin, Ken. Blog. "Does Universal Programming Literacy Even Make Sense?" "Yes, but *Why*?"

In these two blog posts, this SIGGRAPH star computer scientist argues that computer programming should be thought of as writing, but that perhaps we don't have the right language to accomplish that scale of programming just yet. In the second blog post (a response to comments on the first), he comes up with an analogy of programming to cooking, which lots of people perform at home and iterate on. Building on the metaphor, he writes, "the tools that allow millions of people to program in a powerful way are going to be those that allow those people to achieve goals which really matter to them."

Perlin, Ken. Blog. "Does Universal Programming Literacy Even Make Sense?" 24 Feb 2008. Web. http://blog.kenperlin.com/?p=97 . "Yes, but Why?" 27 Feb 2008. Web. http://blog.kenperlin.com/?p=100 .

Work in English Studies

Manovich, Lev. Software Takes Command.

This e-book is available free here, and is a great introduction to software studies and the major ideas and players in this new field. In this book, he's interested in the culture of software, and has moved away from an emphasis on computer science (which he had promoted in the 2001 version of the book). He wants to explore how software is constituting society, and how society constitutes it. "our contemporary society can be characterized as a *software society* and our culture can be justifiably called a *software culture* – because today software plays a central role in shaping both the material elements and many of the immaterial structures which together make up "culture"" (16-17). No literacy discussion per se, but an important work for this area nonetheless.

Manovich, Lev. Software Takes Command. Web. 2008. http://lab.softwarestudies.com/2008/11/softbook.html

Hayles, N. Katherine. *My Mother was a Computer* and "Deeper into the Machine"

Electronic literature scholar Hayles has long been interested in intersections

between code and text from a literary perspective. From a *literacy* perspective, the work is less relevant, but I still find her theoretical approach useful. *My Mother was a Computer* also provides a great review and interpretation of Derrida, Saussure, Hansen, Aarseth, Kittler, McGann, Kurzweil, and Wolfram—all of whom are relevant to code studies in some way. The most interesting moments in this book for me were her look at the "worldview" of speech, text and code, and what each implies. For instance, the "regime of computation," draws on Stephen Wolfram's cellular automata. There's revealing and concealing inherent in code (through objects in OOP, for instance), and that ability is built into our ways of looking at the world now. She then applies this analysis to literariness and wants to see how simple systems can grow more complex for high-level literariness.

Her essay, "Deeper into the Machine," explores the literary "push toward the creation of a creole comprised of English and code. These works draw on the literary tradition and programming protocols to ask what it means for contemporary users to be constructed by both. What kinds of subjects are spoken by this creole?" She asks, "What kinds of subjectivities are implied by the interfaces created by these works, and what is their relation to the machines that write them?" (372). The book *Writing Machines* also explores this idea, with some help from Turing's concepts of computational simulation.

Hayles, N. Katherine. "Deeper into the Machine: Learning to Speak Digital." *Computers and Composition* 19.4 (2002): 371-86. *ScienceDirect.* Web. 19 Aug 2008.

Hayles, N. Katherine. *My Mother Was A Computer: Digital Subjects and Literary Texts*. University of Chicago, 2005.

Selber, Stuart A. Multiliteracies for a Digital Age.

This is an important book for the fields of composition and rhetoric / computers and writing, in particular. It's focused on academic administration, and outlines a program to structure teaching of multiliteracies. Selber's framework of functional, critical, and rhetorical literacies is well thought-out and useful. The most interesting for computer programming and literacy is the rhetorical framework, which focuses on production of compositions with computers, although Selber doesn't deal much with programming as part of this production.

Selber, Stuart A. (2004). *Multiliteracies for a digital age.* Carbondale, IL: Southern Illinois University Press.

Leblanc, Paul. Writing Teachers Writing Software.

Strangely, this book isn't referenced very much in the fields of composition and rhetoric or computers and writing, although it comes from those fields and I think it should be. It's another of those early 1990s "what could have been" scenarios, as Leblanc imagines, well, writing teachers writing software—particularly to support writing practices. This book was before the hegemony of Microsoft Word, and reveals a moment where this appeared a more plausible possibility. His vision and enthusiasm makes this a fun read.

Leblanc, Paul. *Writing Teachers Writing Software*. Advances in Computers and Composition Studies. Urbana, IL: NCTE, 1993. Print.

Dissertations

Black, Maurice. "The Art of Code."

This is an excellent dissertation that is, unfortunately, very hard to get because it only exists in a single, print-copy form that must be requested through the UPenn library. (I don't think the author has ever published from it, but please alert me if so! I think he might have left academia.) Nick Montfort's notes are great (and what persuaded me to request the dissertation to read): http://nickm.com/if/art_of_code.html . My more detailed notes are available online, too: http://www.annettevee.com/blog/2012/06/04/my-notes-on-maurice-blacks-the-art-of-code/

Black, Maurice. "The Art of Code." University of Pennsylvania, Department of English, 2002. Print.

Miller, Jonathan A. "Promoting Computer Literacy through Python Programming."

This dissertation is available on the Python.org website. It's focused on Python, but makes claims about literacy and programming more generally. Miller makes a central claim that computer literacy (by which he basically means programming) builds on traditional literacy—they are "intertwining literacies" (6)—and so working on computer literacy augments rather than competes with textual literacy. This analogy is the most useful for understanding how he frames computer literacy: " "Reading : Writing :: Using a computer : Programming a computer" (10). Miller points out that programming isn't just about writing the code; it's also about conceptualizing the problem, creating a viable algorithm, debugging, expression, etc. Algorithmic thinking is about breaking down a problem into smaller and smaller tasks (50). Miller rests his argument about programming education on this idea: "it will be important to *learn computer literacy* because students will *use computer literacy to learn"* (35). He suggests that programming should be taught across the curriculum because of the thinking and problem-solving it allows people to do, but also because the line between using multimedia software and programming it is blurring. Unfortunately, this is all of Miller's work in this area, as he took his research in other directions afterward.

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

Vee, Annette. "Proceduracy: Computer Code Writing in the Continuum of Literacy."

My dissertation addresses this topic from a literacy studies perspective, with special attention to the history of literacy. I focus on the intersecting technological and social factors of computer code writing as a literacy—a practice I call *proceduracy.* Like literacy, proceduracy is a human facility with an expressive technology. I provide conceptual and contemporary portraits of proceduracy to

place it within a longer social and technological history of literacy, beginning with the transition of text into the infrastructure of English society in the 11th – 13th centuries and continuing through the push for mass textual literacy in the 18th – 20th centuries. I conclude by exploring the pathways to and implications of mass proceduracy. My dissertation isn't available online, but I'll send it to you if you email me. Or, you can just wait for the book!

Vee, Annette. "Proceduracy: Computer Code Writing in the Continuum of Literacy." University of Wisconsin, Department of English, 2010.

Other recommended articles/books

These are all listed alphabetically and I've added notes by many of them. Many of these are important texts, but didn't make it into the list above because they don't deal directly with computer programming as a kind of literacy. Again, a subjective list, and it only contains works that I can recommend personally.

Computer history

Campbell-Kelly, Martin, and William Aspray. *Computer: A History of the Information Machine*. The Sloan Technology Series. 2nd ed. Boulder, CO: Westview Press, 2004. Print. [Great intro text for the history of computing]

Ensmenger, Nathan. "The 'Computer Boys' Take Over." Cambridge, MA: MIT Press. 2010. Print.

---. "Letting the 'Computer Boys' Take Over: Technology and the Politics of Organizational Transformation." *International Review of Social History* Supplement (2003): 153-80. *JStor*. Web. 13 Apr 2008.

---. "Making Programming Masculine." *Gender Codes: Women and Men in the Computing Professions*. Ed. Misa, Tom: IEEE, 2010.

---. "Software as History Embodied." *IEEE Annals in the History of Computing* 31.1 (2009): 88-91. *ACM Digital Portal*. Web. 31 Aug 2009.

Kemeny and Kurtz. *Back to BASIC*. Addison Wesley, 1985. [About the design and early implementation of BASIC at Dartmouth. Readable and interesting, but the authors are bitter about BASIC language design balkanization.]

Light, Jennifer. "When Computers Were Women." *Technology and Culture* 40.3 (1999): 455-83. *Project Muse.* Web. 29 Oct 2009.

Mahoney, Michael. *Histories of Computing*. Cambridge, MA: Harvard University Press, 2011. [Collection of wonderful essays by the late Mahoney.]

Turing, Alan. "On Computable Numbers, with an Application to the Entscheidungsproblem." *The Essential Turing*. 1936. Oxford, UK: Clarendon Press, 2004. 58-90. Print. [Read the birth of the computer!]

Nelson, Theodor Holm. "A File Structure for the Complex, the Changing, and the Indeterminate." *ACM : Proceedings of the 20th National Conference* (1965): 84-100. *ACM Digital Portal.* Web. 11 Jun 2008. [hyperlinks!]

Broad-audience books on history and programming culture

Graham, Paul. *Hackers & Painters: Big Ideas from the Computer Age*. Sebastopol, CA: O'Reilly, 2004. Print. [Graham is the founder of Y-combinator startup promoter and an important thinker in programming. Check out his website here http://paulgraham.com/]

Kohanski, Daniel. *The Philosophical Programmer: Reflections on the Moth in the Machine*. New York: St. Martin's Press, 1998. Print. [Insightful though seldom-cited book on programming philosophy.]

Kushner, David. *Masters of Doom*. Random House, 2004. [about id software founders John Romero and John Carmack, and the early 1990s video game industry]

Lessig, Lawrence. *Code* or *Code 2.0.* [Both are good books and deal with the intersections of law and code, by a popular IP/constitutional law scholar and co-founder of Creative Commons.]

Levy, Steven. Hackers: Heroes of the Computer Revolution. [25th anniversary edition is out now, but first published in 1984. Widely read, fun and important!]

Lohr, Steve. GoTo: […] The Programmers who Created the Software Revolution. Basic Books, 2002. [wonderful history, including of MS Word, PARC, etc.]

Raymond, Eric. *The Cathedral and the Bazaar*. 1999. Sebastopol, CA: O'Reilly, 2001. Print. [Raymond, an open source promoter, coined the major metaphor for open source—the bazaar. Good essays in here.]

Stephenson, Neal. *In the Beginning was the Command Line*. Avon Books, 1999. [Quick and provocative read from the science fiction author who laments the transition from the command line to GUIs.]

Weber, Steve. *The Success of Open Source*. Cambridge, MA: Harvard University Press, 2004. Print. [Really important and readably book about the history and economics of open source software]

Code studies (critical code studies, software studies, etc.)

Berry, David. Philosophy of Software: Code and Mediation in the Digital Age. Palgrave Macmillan, 2011. [Theory-heavy book in which Berry wants to make software more "visible" by unpacking how code runs and how software is built. Also check out his frequently-updated blog, Stunlaw: http://stunlaw.blogspot.com/ , where he's posted articles on iteracy http://stunlaw.blogspot.com/2011/09/iteracy-reading-writing-and-running.html and computational thinking http://stunlaw.blogspot.com/2012/03/computational-thinking-some-thoughts.html

Chun, Wendy Hui Kyong. *Programmed Visions: Software and Memory*. MIT Press, 2011. [Interesting and highly theoretical book on source code as fetish, as memory, and politics of visibility/invisibility.]

Coleman, Biella. *Coding Freedom: The Ethics and Aesthetics of Hacking*. Princeton University Press, Forthcoming Nov 2012. [I haven't read the book because it's not out yet, but Coleman's other work on hackers and code is great, and I'm sure the book will be, too. See her work on her website: http://gabriellacoleman.org/]

DeNardis, Laura. Protocol Politics: The Globalization of Internet Governance. MIT Press, 2009. [Smart and highly technical historical and political analysis of internet governance, through legal and digital code.]

Fuller, Matthew, ed. *Software Studies: A Lexicon*. Cambridge, MA: MIT Press, 2008. Print. [A fun edited collection where each code-savvy author contributed a piece named with some aspect of software, e.g., glitch, Perl, pixel, loop.]

Fuller, Matthew. *Behind the Blip: Essays on the Culture of Software*. Autonomedia: Brooklyn, NY. 2003. [This is a creatively written foray into "software criticism." I have found the first essay—the introduction to the idea—more useful than the others, which are written in a more creative and gestural style.]

Galloway, Alexander. *Protocol: How Control Exists after Decentralization*. MIT Press, 2004. [important theoretical work on how protocol operates, including online through code.]

Galloway, Alexander and Eugene Thacker. *The Exploit: A Theory of Networks*. MIT Press, 2007. [how people can exploit protocol to get into networks, highly theoretical and technical.]

Kirschenbaum, Matthew. Mechanisms: New Media and the Forensic Imagination. MIT Press, 2007. [More about hardware and literature than software, but great and clearly written text that takes the materialist perspective of book history to computers.]

Kittler, Friedrich. "There is no software." *Ctheory*, 1996. http://www.ctheory.net/articles.aspx?id=74 [about intersections/ interactions/ invisibilities between hardware and software, from a philosophical perspective]

Marino, Mark. "Critical Code Studies." *Electronic Book Review* (4 Dec 2006). Web. 30 Jun 2009. <http://www.electronicbookreview.com/thread/electropoetics/codology>. [Where the term "critical code studies" was launched—a literary analysis perspective on reading code.]

Mateas, Michael, and Nick Montfort. "A Box, Darkly: Obfuscation, Weird Languages, and Code Aesthetics." *Proceedings of the 6th Digital Arts and Culture Conference, IT University of Copenhagen* (2005): 144-53. *NickM.com.* Web. 1 Mar 2009. [Wonderful fun article on languages like Brainfuck and the Underhanded C programming contest.]

Shirky, Clay. "Situated Software." *Clay Shirky's Writings about the Internet* (2004). Web. 15 Jan 2010. <http://www.shirky.com/writings/situated_software.html>. [Where Shirky argues about "downsourcing" software to other fields rather than "outsourcing" it]

Wardrip-Fruin, Noah. *Expressive Processing: Digital Fictions, Computer Games, and Software Studies*. Cambridge, MA: MIT Press, 2009. Print. [Great book on programmatic processing in creative works. He argues that analysts of this work need to understand the code to understand the work.]

In composition and rhetoric

Ballentine, Brian. "Hacker Ethics & Firefox Extensions: Writing & Teaching the 'Grey' Areas of Web 2.0." Computers and Composition Online, 2009. http://www.bgsu.edu/cconline/Ed_Welcome_Fall_09/compinfreewareintroduction.htm

Banks, Adam. Race, Rhetoric, and Technology. Lawrence Erlbaum, 2006. [Awesome intro reference to BASIC; the rest of the book deals with critical issues of race and technology, although less directly with code. Chap 2 has been the most useful to me, personally, and I've assigned it to students.]

Cummings, Robert. "Coding with Power: Toward a Rhetoric of Computer Coding and Composition." *Computers and Composition* 23 (2006): 430-46. *ScienceDirect*. Web. 20 Apr 2010. [Argues for writing teachers to embrace coding.]

Rieder, David. Scripted Writing. Small Tech [He's talking about new trends in “digital writing”, where lines between text and code are blurred, and code is used for poetic potentiality. He asks, “How....do we develop a study of digital writing that recognizes interplay and upholds the distinction between the two species?” (86) His answer, and a main point of the text is to offer poetry as a potential space for these creative acts. For Rieder, code is written for humans primarily—the poetic ways of expressing code in both its text and object form.]

Sorapure, Madeleine. “Text, Image, Code, Comment: Writing in Flash.” *Computers and Composition* 23 (2006): 421-29. Print. [Great article from one of the most creative and technical talents in the field.]

Olson, David R. “Computers as Tools of the Intellect.” *Educational Researcher* 14.5 (1985): 5-8. *JStor.* Web. 29 Dec 2008. [makes some similar arguments to Papert]