Introduction to "The Role of Computational Literacy in Computers and Writing"

Mark Sample, **George Mason University**

Annette Vee, **University of Pittsburgh**

Enculturation: http://enculturation.net/computational-literacy

(*Published: October 10, 2012*)

If we can measure the significance a new scholarly object by the number of innovative courses, breadth of engaging research, and buzz of online activity, then code has reached a critical moment in writing studies. Scholars such as N. Katherine Hayles and Espen Aarseth have long focused on the function of code in electronic texts, but ever since Mark Marino described the humanistic reading of code as "critical code studies" in 2006, the field has exploded. Often working at the level of code, the computer science scholars Michael Mateas and Noah Wardrip-Fruin are exploring games as narratives. Rhetoricians who program commercially such as Ian Bogost are similarly concerned with procedurality in new media, particularly game code. And works such as Bradley Dilger and Jeff Rice's edited collection *From A to <A>* foreground code as a mode of writing. Of course, this work builds on early research by Paul Leblanc, Gail Hawisher, Cynthia Selfe, Jim Kalmbach, and Ron Fortune, all of whom endeavored to draw attention to the politics and composition of code in the 1980s and 1990s.

Code has not only made its way into our research, it has also found its way into our classrooms. Kevin Brock's Code, Computation and Rhetoric class at North Carolina State University, Jamie Skye Bianco's Composing Digital Media course at University of Pittsburgh, and James Brown Jr.'s Writing and Coding composition course at University of Wisconsin-Madison are all examples of writing classrooms that now include code. In programs like Communication, Rhetoric and Digital Media at NC State, there is a growing recognition that some expertise in computational thought is a part of future success as a scholar affiliated with the humanities. As David M Rieder has argued, code is blurring into our textual compositions, such that it's no longer possible to bracket it off from writing pedagogy.

It may be that composition and rhetoric teachers are discovering what software developers already know. That is, from the perspective of computer science, programming has long looked like writing. Turing Award-winner Donald Knuth has argued for argued for "literate programming" and Frederick Brooks has drawn a parallel between a parallel between the programmer and the poet. More recent efforts under the name of "computer science for all," "computer programming for everybody," or "computational thinking"—promoted by computer science educators such as Mark Guzdial and Jeannette Wing—have sought to teach computing across the university curriculum, just as writing is now. These educators build on decades-old efforts by Seymour Papert (the designer of Logo) and John Kemeny and Thomas Kurtz (the creators of BASIC), who designed languages that would help a wider spectrum of people learn to code.

Beyond academia, popular initiatives and news stories have made the connection between coding and writing. For example, the Code Year initiative made a big splash in January 2012 with some good PR, as well as a promise by New York Mayor Michael Bloomberg that he would learn code this year; as of this writing, over 450,000 people have signed up to receive weekly code lessons via email. Massively Open Online Courses, or MOOCs, taught by professors at prestigious institutions such as Stanford and MIT have inspired tens of thousands of civilians to enroll in basic computer science courses. Writing for The Guardian in March 2012, John Naughton explains "Why all our kids should be taught to code." Media theorist Douglas Rushkoff caused a stir in 2011, arguing, "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."

It is against this backdrop that the Town Hall “Program or be Programmed: Do We Need Computational Literacy in Computers and Writing?” was proposed. The panel organizer, David Rieder, saw the growing interest in software studies and critical code studies, the ever-deepening engagement with computational approaches to the digital humanities, and his own local experiences teaching graduate courses on Arduino and Processing add up to an emerging interest in code and writing studies. Rieder asked Annette Vee to help plan and recruit panelists for the Town Hall, and both were delighted that writing and coding scholars Alexandria Lockett, Elizabeth Losh, Mark Sample, and Karl Stolley agreed to speak.

In this edited iteration of the Town Hall (you can watch the original on Vimeo, courtesy of Dan Anderson), we’ve provided a more polished version of each panelist’s script, along with a few sample tweets from the audience to represent the dialogue with the presenters. This Storify

captures a more complete Twitter backchannel during the presentation. Lynn C. Lewis's review review for the University of Michigan's Sweetland Digital Rhetoric Collaborative offers additional context for the original presentations.

As the title of the Town Hall implies, the pieces below are meant to be provocative rather than thorough considerations of the role of code and computational literacy in computers and writing. Rieder begins by arguing against the logocentric bias of writing studies, and asks: why limit texts to their readability? Based on their power and ubiquity, we should understand processes, specifically algorithmic ones, as the new basis of writing. Vee picks up on Rieder's final assertion that those who don't code will be "stuck in the logocentric sands of the past" and breaks down what it means to have "coding" defined by a coterie of specialists. She asks: if coding is central to writing, shouldn't the values associated with good code be as diverse as the values associated with good writing? Yet we cannot ignore the social context already established for code, asserts Sample. Invoking historically saturated statements from the BASIC programming language, Sample outlines concerns about coding style, substance and interpretation to indicate code's social nature. The social nature of code is clear for Lockett, who relates coding culture to code-switching across versions of English and notes the "sponsorship" of a friend with whom she exchanged knowledge about political and rhetorical theory for knowledge about code. The tension between her assertion that she is "not a computer programmer" and her facility with Linux and various coding structures resolves—at least provisionally—in her embrace of the term "hacker," which gives her the latitude to argue for political awareness about the "socio-technical systems" associated with code and software. Finally, Stolley gives us a call to arms: in Computers and Writing, we must become familiar with the tools that define our field. It will not be easy to learn Unix, GitHub, Ruby, and Javascript, but Aristotle argued that "people become builders by building," Stolley reminds us, and ease of acquisition should not dictate what skills we choose to learn.

Responding to these various assertions, Losh reflects on what it might mean to teach and learn code within the context of Computers and Writing. To the anxious instructor struggling to teach students the complexities of writing, much less programming, Losh advises that we think of writing as an "informational art." This perspective can widen the approaches we take as well as the modalities we emphasize in our classes. But to the administrator who must now direct departments to account for programming's role in writing, she offers only tentative models rather than easy answers. Although we ultimately cannot provide an answer to the central question of the panel—"What is the role of computational literacy in computers and writing?"—we hope our provocations contribute to the continuing conversations about coding and composing in writing studies.

David M. Rieder: Programming Is the New Ground of Writing ›