“User-Friendly Classrooms: Interconnections Between Today's Media Composition Software and the Unusable Computers of the Early 1980s”

Michael L. Black, University of Massachusetts-Lowell

Presented at The Twelfth Biennial Thomas R. Watson Conference on Rhetoric and Composition
University of Louisville, October 25-27, 2018.

In teaching my students how to explore the expressive affordances of digital composition and content creation technologies, I inevitably also teach them how to use specific software applications and online publishing platforms. Working with digital media requires engagement with user-friendly software applications and user-friendly software platforms in the sense that any digital content we create has to be algorithmically structured and published somehow. In today’s media ecology, there is also a sense that publishing and promoting the circulation of media texts online is an important part of their rhetorical construction. This attitude is especially true in the institutional context that I teach in: a professional writing program. For my students, learning to leverage user-friendly software like Adobe Creative Suite or platforms like Facebook, Wordpress, Google, YouTube, Twitter, and Instagram are important aspects of the digital literacy they seek to cultivate. Regardless of how they may feel about this particular applications and platforms, students understand that familiarity with them is something that employers value—and indeed the perception that we are bringing professional expectations into our classrooms has been a big part of how our department promotes our writing courses. Yet, as I want to discuss today, it’s also important to recognize that the way these user-friendly tools encourage us to understand our relationship to computers in overly simplistic terms, without recognizing that the commonplace attitudes about how computers should be used that they reinforce are the product of a complex network of discursive influences that can be traced back to at least the 1970s.

What I want to discuss, in other words, is a tension between what we tend to teach as "digital literacy" and what people in the software industry and academic human-computer interaction research call "usability." Over the past twenty years, researchers in the humanities, and particularly those who study writing, have come to define digital literacy as a set of skills necessary to communicate using computers. The work of Cynthia Selfe, Stuart Selber, Annette Vee, just to name a few, shows that what counts as digital literacy is subject to change and has been defined largely as the computer skills that employers believe skilled workers should possess. Interestingly, as Vee notes, that has led to a resurgence of programming being considered a core competency in fields other than computer science.

And while that is certainty true in some professions, minimizing exposure and user engagement with explicitly technical processes like programming has been and continues to be a key goal for developers of modern operating systems and consumer devices since the early 1980s. Let's begin moving backwards by briefly looking at usability, as defined by Donald Norman in 1986, around the time that human-computer interaction emerged as a distinct discipline. Whereas in the humanities, our study of digital literacy is often part of a larger project of engaging with the cultural and social issues surrounding computing in and beyond our classrooms usability seeks to avoid engaging with those issues. Early theories of personal computer usability were proposed by researchers working in industrial engineering, artificial intelligence, and cognitive psychology, and they were concerned with describing how people built mental models of computer systems during use. Here, Norman notes that there are three models at play in a given interaction: the model the designer has of the software's intended use, the model as expressed in the software itself, and the model that the user constructs in their mind. Software designers, he explains, really only have control over the first two of these models. They can't control for the identities, experiences, knowledges, or interests of the people that user their software. Ultimately, in human-computer interaction theory, this heterogeneity is a problem to be solved. Offering users simplified engagements with teaching, Norman and other argue, eliminates room for error and misinterpretation, meaning that designers can imagine a universalized, generic user rather than account for the range of identities of their users. While I think some areas human-computer interaction today have done well in addressing the diverse needs of users—particularly those researchers who work on interfaces for people with disabilities—usability theory with respect to software design has since largely carried forward dehumanizing assumptions like this one.

And yet the same time, these principles are wrapped up in a rhetoric of user-friendliness, which promises to "humanize" computing so that the computational practices the present us with will mesh more readily into our everyday lives. Today's user-friendly software is said to empower users, allowing us to connect and communicate in ways that were never before possible, all at the push of a button. As Kristin Arola and Kory Lawson Ching note, they do so largely through narrowing and automating our communication practices. We no longer "write web pages," for example. Instead, web design often takes the form of filling in forms or adjusting settings in a template. User-friendly software simplifies our relationship to computer systems by requiring us to know less about how they work, offering us singular models of communication and requiring that we accept those models as a condition of use. And as Siva Vaidhyanathan, Frank Pasquale, and Safiya Noble have shown, it's not just algorithms and data structures that are hidden from us, but also the political assumptions embedded, intentionally or otherwise, into their design. Those, too, we accept as a condition of use when we use user-friendly software or publish on user-friendly platforms. And here, in short, is my concern: the way that the digital literacies we teach and study are bounded, structured, and defined by technologies produced according to this kind of usability theory. They naturalize specific ways of thinking about our relationship to computers, making computer systems largely invisible in our discussion of them. User-friendly software encourages us to talk about how to do things with computer systems, but not, in specific terms, what those computer systems are doing.

What I've found, looking back through discussions about personal computing in the 1970s and 80s, is that usability as user-friendliness emerges in response to widespread anxieties about digital literacy, then called "computer literacy." When the idea of "personal computing" first appeared in the 1970s, it was both a commercial label and a model of computer use that was explicitly political in nature. You can see this political framing in Joy Rankin's new book, or in Ted Nelson's self-published Computer Lib from 1974. The idea was that people needed individual, unstructured access to computers to see for themselves that they were not just tools for technocratic control: that they could be put to creative purposes. As Nelson and others argued, we needed a grassroots computer literacy movement that allowed ordinary people to stand up to what he called the "computer priesthood." This priesthood, he noted, was fond of responding to criticism about the cultural and social effects of computer systems with the phrase "that's just not how computers work." If we all understood how they worked, he argued, then not only could our criticisms of technologists become more pointed, but their dismissive rhetoric would lose its power.

By the 1980s, the idea of a "personal computer revolution" had made its way into the popular press. But its depiction here didn't exactly match the grassroots literacy and creative computing envisioned by the personal computer revolutionaries like Nelson. Instead, the popular press was focused almost exclusively on business uses and computers in office settings. You can see that in feature stories like this one from US News and World Report, or others published in Time or Newsweek, which gush about how they automate tasks that get in the way of the "real work" and open up new perspectives on financial data.

When trying to discuss other uses for personal computers, articles typically offered the same tired refrain: they were good for managing your taxes, keeping track of recipes, using edutainment software, and accessing an as yet non-existent internet. Whereas general interest and news publications were excited about computers, many tech journalists in the early 1980s were highly skeptical about this idea of a computer revolution. In this 1981 editorial from the Technology Review, for example, the author notes that realizing the paltry benefits of non-office uses required a great deal of time and energy: you could do the things they offered with less expense and less effort without a computer. The problem, he notes, is that personal computers "still reek[ed] of their computer science origins." They were baffling to people who didn't already understand how computers worked. They weren't meant for the average person. At least not yet. As this journalist opines in a later column, Guternberg suffered from a similar problem, "it took widespread literacy for the printing press to make its impact. . .The home computer awaits a similar rise in computer literacy."

In fact, some tech journalists even went so far as to claim that we were on the verge of a "computer literacy crisis." Computers were axiomatically seen as good for business and being foisted upon us without any real commitment to training and education. Outside the office, computers were seen as remote from our everyday concerns. As these cartoons show, intense frustration with them was becoming commonplace—part of the normal experience of learning to use one. And once you learned to use one, you became removed from everyone else, part of some subculture that only a few people really understood. While more and more people were using computers with each passing year, increased interest was accompanied by a sense of anxiety that not everyone would be able to learn how to use one.

And so, some journalists began to ask, maybe we needed to rethink what we meant by "computer literacy." The headline of this magazine isn't suggesting that people don't need computer literacy, but rather that we didn't need computer literacy as it was currently understood. Wouldn't it be better if people could focus on other things, besides the computer itself, when using them? In the early 1980s, many journalists looked forward to a future when we talked about we could do with computers, not about the technical aspects of computers. Many confidently believed that the threat of a computer literacy crisis was overblown. Ultimately, such a crisis would be avoided because there was clearly a market demand for a new kind of computer that supported a new kind of computer literacy.

Enter, the Macintosh. Apple Computer had, of course, been around since the late 1970s, but by the early 1980s the popular press frequently characterized the company as stagnating. In promoting this new computer, Steve Jobs drew on these discussions about a new kind of computer literacy and offered a framework that usability theorists would later adopt as the goal of design. He notes here in an interview that no one has time to learn how a computer works, especially not the office workers we're asking to sit down in front of them everyday. We view the Macintosh nostalgically as a creative tool, but it's important to remember that its software was first developed for an earlier sytem, the Lisa, which was explicitly designed to be marketed within a different side of the computer industry: “office automation.” The rhetoric Jobs employs meshes the concerns I’ve outlined moments ago with claims made in advertisements from other office automation companies like Lanier and Wang.

And here, really, is the goal of this approach to design: “When you use the Macintosh, there is no such thing as an operating system. You never [see yourself as] interact[ing] with it: you don’t know about it. Users are much more concerned about what the computer will do, what it will communicate with, which is the right way of looking at products.” This idea, that we could use computers without thinking about them as computers, would soon be picked up academic usability theorists like Ben Shneiderman, Terry Winograd, Donald Norman, and Brenda Laurel. If I had more time, I go into detail about how developments in academic usability theory tended to follow successes within the personal computing industry, but that is probably best saved for a later conversation.

In summary: key assumptions within usability theory, the frameworks that define user-friendly design, emerged in response to fears of a computer literacy crisis during the 1980s. The Macintosh’s legacy is best understood as one that redesigned personal computers in ways that redefined computer literacy, narrowing and refocusing the skills associated with it away from those believed to be too technical for most people to understand. Now, we can talk about what we want to do with computers without having to understand what computers do, ostensibly on our behalf and for our benefit. This is the user-friendly future, imagined in the 1980s, that we now live in. But I wonder, are we beginning to live out a computer literacy crisis of our own? It’s not the same, to be sure, but many of the recent scandals and major tech-problems revolve around the fact that no one, except for a small number of corporate developers, understand the technologies we’ve integrated into our everyday communication practices. I’m not yet sure the best way to represent this problem to my students in a way that will drive, rather than discourage, their rhetorical exploration of personal computer technologies. But I hope to model in my assignments a digital literacy that encourages them to value complex engagements with computation rather than accept usability's insistence that only simplified interactions can be useful.