Chapter 14
We Can and Must Understand Computers NOW

Noah Wardrip-Fruin

14.1 Three Phrases

From the endlessly quotable Ted Nelson—whose neologisms pepper the language we use to understand the present, from "hypertext" to "visualization"—perhaps no phrase is better known than, "You Can and Must Understand Computers NOW." It was emblazoned across the *Computer Lib* side of his 1974 *Computer Lib/Dream Machines* (*CL/DM*), the most influential book in the history of computational media.[1]

Nelson's call is not only memorable today, but still quite relevant. For example, consider the recent revelations of massive government surveillance, as disclosed by Edward Snowden and others. Without a deep understanding of computing, one might debate whether the vision of Total Information Awareness is morally right, or is instead sending us down a path to an "Orwellian," *1984*-style future. However, with a deep understanding of computing, one can not only raise the questions of morality in more depth, but one can also see that Total Information Awareness is a technically unworkable fantasy (like the Star Wars program pursued by the Reagan administration in the non-fictional 1980s) providing a false rationale for treating everyone as a suspect.

In other words, one reason that we must understand computers now is so that we can understand what is happening, and make informed choices, as members of a computationally-steeped democracy. We need to understand computing so that we can see past deceptions about what computers can do, and how computers work. As

[1]For example, as Steve Wozniak said at Intertwingled, "At our computer club, the bible was *Computer Lib*" — referring to the Homebrew Computer Club, from which Apple Computer and other major elements of the turn to personal computers emerged [18].

N. Wardrip-Fruin (✉)
Department of Computational Media, University of California, Santa Cruz,
1156 High Street, MS:SOE3, Santa Cruz, CA 95064, USA
e-mail: nwf@soe.ucsc.edu

D.R. Dechow, D.C. Struppa (eds.), *Intertwingled*, History of Computing,
DOI 10.1007/978-3-319-16925-5_14

Nelson puts it colorfully, “Down with cybercrud!” However, that is not the only reason we must understand computers.

In thinking about the other reasons we must understand computers, and in considering a variety of projects and ideas that have sought to help an understanding of computing become more widespread, I believe we should also attend to two further phrases from *CL/DM*—neither as well known, but each extremely telling. The first, also from the 1974 edition, is, “presentation by computer is a branch of show biz and writing, not of psychology, engineering or pedagogy” [10, DM2]. The second, added in the 1987 edition, is, “All Simulation is Political” [11, CL149].

I choose these two additional phrases, in part, because they point to ideas of Nelson’s that have deeply shaped my own thinking and career—a career in computational media that came into focus after I found a copy of *CL/DM* in my college bookstore. A piece like this one could be written about other facets of my thinking, using a different selection of *CL/DM* phrases, and I believe the same is true for many of the most insightful people I’ve met in the field—that their thinking was indelibly shaped by an early encounter with Nelson’s ideas. But for telling this story, let me begin with “You Can and Must Understand Computers NOW.”

14.2 We Can and Must

Nelson is certainly not alone in calling for broad understanding of computing, in some form, and not the first to do so. The earliest example I can find is Alan Perlis’s call—in 1961—for all university Frosh to take a programming class [12]. This is pretty obscure. A much better known example is the Logo project (often remembered for its “turtle” graphics) created by Seymour Papert, Wallace Feurzeig, Daniel Bobrow, and collaborators beginning in 1966 [2]. A more recent example is Jeannette Wing’s call for broad “computational thinking,” which she characterizes as a set of conceptual tools for “solving problems, designing systems, and understanding human behavior” [17].

These sorts of undertakings are generally noble projects. But I would argue that, at root, many aren’t actually about people understanding computers (it is closer to a side effect) and they certainly aren’t about what Nelson is calling for. In “Logo: A Project History” Anit Chakraborty, Randy Graebner, and Tom Stocky write, “the original Logo developers were out to change mathematics by helping children improve problem solving abilities” (1999). Similarly, as a 2012 report from the UK’s Royal Society notes, computational thinking is primarily about thinking like a computer scientist in a wide variety of contexts, rather than understanding computers in Nelson’s sense.[2]

[2] “Computational thinking is the process of recognising aspects of computation in the world that surrounds us, and applying tools and techniques from Computer Science to understand and reason about both natural and artificial systems and processes” [5].

14.3 Show Biz and Writing

In the introductory pages of *Dream Machines,* Nelson makes it clear why *CL/DM* is a book with two sides. He does not aim for broader understanding of computers, through the information found in *Computer Lib,* simply because our society is becoming more computational in general. Rather, as he writes:

> My special concern, all too tightly framed here, is the use of computers to help people write, think, and show. But I think presentation by computer is a branch of show biz and writing, not of psychology, engineering or pedagogy. This would be idle disputation if it did not have far-reaching consequences for the designs of the systems we are all going to have to live with. [10, DM2]

In other words, we all must understand computers not just because computers are important, but because the media of the future (and now the present) are computational. We need people who are able to understand, work in, and invent computational media—media that, in Nelson's words, continue the traditions of "literature, film and scholarship"—and are able to do so with, "art, zest, intelligence, and the highest possible ideals" [10, DM2]. This is very much not the same thing as thinking mathematically, or thinking like a computer scientist.

Luckily, there is a tradition of work that takes media and literacy more seriously. The Smalltalk programming language was developed in the 1970s by Alan Kay, Dan Ingalls, Adele Goldberg, and others [6]. Together with the vision of the Dynabook personal computer, it presented an approach to computing that focused on reading and writing (that is to say, computational literacy) and the creation of media and media-making tools (including simulations). And a number of the descendants of Smalltalk and Logo are concerned with media-making and broadening literacy, such as the Processing language for artists and designers, the Scratch language that uses snap-together tiles, and the games-focused Kodu language [7, 13, 14]. There is also conceptual work that seems to foreground literacy issues, as seen in the "computational literacy" discussed in Andrea diSessa's book *Changing Minds* [3].

But here, as with much else in *CL/DM,* Nelson's warning proves prescient. While what we need is a convergence of computing with the arts and humanities, what we get is more often "psychology, engineering or pedagogy." diSessa's book, for example, is primarily concerned with science education, rather than literacy as understood in the traditions of literature, film, and scholarship. More broadly, much of the Human-Computer Interaction community seems convinced that compelling computational media forms can be discovered and designed through psychology-style experiments. Attempts to move computational media forward through pure engineering approaches, in areas such as computer graphics, give us awful "photorealistic" films such as 2007's *Beowulf*—while those few who understand that computing and art must work together (that high-level technical goals cannot be set or evaluated apart from artistic goals) create much stronger, more stylized animations such as 2008's *WALL-E.*[3]

[3] "Computational Media" has recently emerged as a name for the type of work that performs this interdisciplinary integration [15].

Happily, some of the work that follows Logo and Smalltalk does come from those who understand these issues, and are themselves media makers. Ben Fry and Casey Reas, the initiators of Processing, are an accomplished artist and information designer. Matt MacLaurin and Stephen Coy, key creators of Kodu, are both game industry veterans. Projects like these succeed at creating media-centric environments that broaden the ability to understand computers and make computational media—and they do so by embedding media-making knowledge from their creators into computational structures.

But that is not all that is embedded.

14.4 All Simulation Is Political

In the 1987 edition of *CL/DM,* Nelson adds a section with the headline, "All Simulation Is Political." He writes below it:

> Every simulation program, and thus every simulation, has a *point of view.* Just like a statement in words about the world, it is a *model* of how things are, with its own implicit emphases: it highlights some things, omits others, and always simplifies. The future projections made by a simulation only project those views forward in time. [11, CL149]

In the kinds of media made with Kodu and Scratch (and many other related systems, such as Alice, Squeak, and AgentSheets) *simulation* is a primary form of representation. The world is represented through rules and the interaction of (and our interactions with) those rules over time, together with data that represents the world state and constants.

As Nelson observes, all the simulations created with these systems embed assumptions about the world that derive from viewpoints—they are political. Similarly, systems for creating simulations are also based on assumptions, derived from viewpoints.

In an individual simulation we see the politics in the rules and data. In a system for creating simulations we see the politics in the available elements and process of creation. In both cases, the politics are often implicit and unconscious.

Consider the Kodu system.[4] Kodu focuses on making games, and has a model of agent-oriented programming (using robotics-style sensors and actions) that can be carried out with an Xbox controller. Almost everything is menu driven, and many problems that plague beginning programmers (such as syntax errors) are effectively eliminated by the system's approach. By programming different agents to interact with each other and the environment, an autonomous simulation can be created. But Kodu's tutorials focus instead on game projects, leaving one or more agent(s) under the control of a player. Everything created is rendered in smooth 3D, often using professional models and textures included with Kodu—creating a sense of polish

[4] Kodu is both an influential system itself and the basis of Microsoft's Project Spark, launched in October 2014.

for even the simplest project, and making even the sculpting of the environments in which agents interact an appealing activity for many in the target age group (roughly 7–14, though with an emphasis on the upper end of the range).

In a menu-based system such as Kodu, perhaps the simplest way to surface some of its assumptions, and therefore its politics, is to look at the menu structure. Kodu's menus are hierarchical, with the elements on the top level the fastest to discover and use, presumably reflecting assumptions about what will be most useful. Here are some observations about Kodu's menus:

- Shooting is one of a handful of actions in the top-level menu.
- Being shot is one of a handful of sensors in the top-level menu.
- Saying something is in a sub-menu.
- No menu items support an internal life for characters, or social relationships between characters.

We might ask ourselves, does this really reflect the range of what a diverse group of early teenagers would care about, and want to represent about the world? Of course not. It reflects particular interests—and the male-dominated subgroup most interested in them is not one underrepresented in computing. In other words, despite the nobility of the project, the implicit politics of Kodu's menus of actions and sensors is the politics of the status quo—shaping what can be said, and who can say it, along familiar lines. In this it is far from alone.

14.5 Understand Computers How?

While in theory we could create computational media about anything that we could write about, or make a film about, in practice our tools and established genres generally support a much narrower range. In a sense we live in the world Nelson warned about, in which the designs of the systems we live with do not support broad thinking, expression, and innovation.

But I write about Kodu in this chapter because, when I was part of a group that approached the Kodu team, we found a genuine interest in shifting its expressive range. We worked primarily with Matt MacLaurin, Brad Gibson, and Kent Foster at Microsoft (Kodu emerged from Microsoft's FUSE Labs and Microsoft Research). Our team included Teale Fristoe, Jill Denner, Michael Mateas, Brandon Tearse, Larry LeBron, Eric Kaltman, and Gina Lepore (all from UC Santa Cruz or ETR Associates).[5]

We did some simple things, like adding an easy way for characters to listen for language (not just speak it) which became part of the main Kodu distribution. But

[5] The first stage of our work is described in "Say it With Systems" [4]. The project was supported in part by the National Science Foundation (under Grant No. DRL-1042944). However, any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

we also experimented with more complex changes, such as giving characters different levels of friendship, and creating sensors and actions that made it possible for agents to alter and respond to these levels. We worked to make these almost as simple to use as those for shooting and being shot, and we rearranged the menus so that they were at the same level. We created new curricula, introducing Kodu in new ways, and new sample games, emphasizing our new sensors and actions. We did all this in the context of talking with early teens from a variety of socio-economic backgrounds, and we ran after-school programs in a variety of middle schools, using versions of Kodu iterated between each program.

What we found, perhaps unsurprisingly, is that by the end we were seeing a much wider variety of games. There were more types of play, and a broader range of subject matters. In fact, what we eventually found was that Kodu's visual polish—an important part of its initial appeal—became one of the barriers. Its models and animations were created for a system that made shooting and racing gameplay easy. When teens started considering making a broader set of games, they saw a mismatch between their potential game systems and the ways the games could appear on screen. But in a sense, when teens observed this it was also positive. It was the beginning of a critique of the assumptions built into Kodu's available elements, opened by shifting the available rules without shifting the data.

To my knowledge, what we did with Kodu has not been done with any other tool. Current tools and ideas may aim to broaden understanding of computing, they may focus on computational media literacy, and they may embody lessons learned from media making. But they are divorced from critical thinking about their representations—from point of view, from politics. They haven't been critiqued regarding the way their technical specifics connect to ideas of the world, much less reshaped in response to critique.

When this is how we do work in our field, we run a significant risk. We run the risk that all these well-intentioned projects end up solving precisely the wrong problem. Nelson did not say that we can and must understand computers because the IT sector (or the surveillance state, or Walmart) has an urgent need for more computing-literate worker bees. Nelson's challenge is only answered if we educate people who are prepared to disrupt business as usual—and to invent the broad, thoughtful media of the future.

14.6 Reading and Writing

Putting Nelson's three statements together, we see an urgent call for a creative and critical literacy of computing broadly, and computational media in particular. This call is as pressing today as it was when *CL/DM* was first published.

Thinking in terms of critical literacy also reveals something rather odd about most attempts to broaden understanding of computing. They are almost entirely focused on *writing,* on the construction of computational artifacts (whether through textual code, Kodu menus, Scratch blocks, or some other means). But this is not the approach we use with other forms of critical literacy. We don't assume, for example, that someone who is going to "read and write" the language of cinema should be concerned solely with shooting and editing their own films, never watching and critically interpreting existing films.

Of course, there are those who have addressed, or at least identified, this gap. Michael Mateas's call for "procedural literacy" is an early call for a critical literacy for computational media makers [8]. Ian Bogost's "procedural rhetoric" draws on the history of rhetoric for a model of critically understanding and making processes [1].

And in recent years work on critical interpretation of computing, taking the technical level seriously, has blossomed. The MIT Press has been one of the leading supporters of this, initiating new book series in both software studies and platform studies. However, this critical *reading* generally still remains divorced from *writing*. I know of no educational institution that teaches them together (e.g., no introductory programming course that includes introductory software studies content) and I know of only one published scholarly book that includes the writing of software as one of the critical methods it uses in analyzing software (the unusually-titled *10 PRINT CHR$(205.5 + RND(1)); : GOTO 10* [9]).

Of course, while undertakings such as software studies seem new to many, for those of us who read Nelson's work it is simply the continuation of his tradition. *CL/DM* contains much that is clearly the critical interpretation of software, connecting the technical level to the cultural one—ranging from discussing the "drill and practice" assumptions built into the TUTOR programming language to exposing the simple workings of *Eliza* and other systems used to market artificial intelligence ideas [10, DM27, DM14]. It is heartening to see the continuation of this work finally being taken up by a wider group, and we can only hope that it is increasingly brought together with attempts to broaden the writing side of a creative and critical computational literacy.

14.7 Conclusion

I'm deeply honored to have the opportunity to contribute to this volume, just as I was honored to have the opportunity to help bring Nelson's writing to a new generation when he gave permission for sections of *CL/DM* and other texts to be reprinted in *The New Media Reader* [16]. While Nelson's work is certainly of historical importance, it also has much to tell us in the present—providing a necessary perspective for evaluating what we are doing in the field, and pointing in directions of great importance for us to pursue. I hope that this chapter provides a useful example of one way this has been done.

I also hope that the broader implications of the lessons I draw from selecting the three highlighted *CL/DM* phrases are clear. To put them another way: If we educate everyone to think creatively and critically about and with computational media, we will also be educating them to think critically about computing—to read simulations for their biased assumptions, to know that warrantless wiretapping of every citizen is not only wrong, but pointless, and more. And that, I believe, is the way in which *we can and must understand computers now.*

References

1. Bogost I (2007) Persuasive games: the expressive power of videogames. MIT Press, Cambridge, MA
2. Chakraborty A, Graebner R, Stocky T (1999) "LOGO: a project history." Website for the structure of engineering revolutions, Mindell DA (ed). December 1999. http://web.mit.edu/6.933/www/LogoFinalPaper.pdf
3. DiSessa AA (2001) Changing minds: computers, learning, and literacy. MIT Press, Cambridge, MA
4. Fristoe T, Denner J, MacLaurin M, Mateas M (2011) Say it with systems: expanding Kodu's expressive power through gender-inclusive mechanics. In: Proceedings of the 6th international conference on foundations of digital games, ACM, pp 227–234
5. Furber S (2012) Shut down or restart? The way forward for computing in UK schools. The Royal Society, London
6. Kay AC (1996) The early history of Smalltalk. In: History of programming languages—II. ACM, pp 511–598
7. MacLaurin, MB (2011) The design of Kodu: a tiny visual programming language for children on the Xbox 360. ACM Sigplan Notices 46(1):241–246. ACM
8. Mateas M (2005) Procedural literacy: educating the new media practitioner. On Horiz 13(2):101–111
9. Montfort N, Baudoin P, Bell J, Bogost I, Douglass J, Marino MC, Mateas M, Reas C Sample M, Vawter N (2012) 10 PRINT CHR $(205.5+ RND (1));: GOTO 10. The MIT Press, Cambridge, MA
10. Nelson TH (1974) Computer lib: dream machines. Self published
11. Nelson TH (1987) Computer lib: dream machines. Tempus Books of Microsoft Press, Redmond
12. Perlis AJ (1961) The role of the digital computer in the university. Comput Autom 10(4 &4B):10–15
13. Reas C, Fry B (2006) Processing: programming for the media arts. AI Soc 20(4):526–538
14. Resnick M, Maloney J, Monroy-Hernández A, Rusk N, Eastmond E, Brennan K, Millner A et al (2009) Scratch: programming for all. Commun ACM 52(11):60–67
15. Wardrip-Fruin N, Mateas M (2014) Envisioning the future of computational media: the final report of the media systems project. Center for Games and Playable Media, UC, Santa Cruz
16. Wardrip-Fruin N, Montfort N (eds) (2003) The new media reader. MIT Press, Cambridge, MA
17. Wing JM (2006) Computational thinking. Commun ACM 49(3)
18. Wozniak S (2014) In "Intertwingled: afternoon session #2." Chapman University, Orange, California. Video timecode: 58:14. http://ibc.chapman.edu/Mediasite/Play/52694e57c4b546f0ba8814ec5d9223ae1d