

Z Y X W V U T S R Q P O N M L K J I H G F E D C B A

[nickm.com](http://nickm.com) > [interactive fiction & digital media](#)

# "The Art of Code" by Maurice J. Black

## Reading notes by Nick Montfort

*Black, Maurice J. 2002 "The Art of Code." PhD Dissertation, University of Pennsylvania, Department of English.*

These are my notes from reading this intriguing dissertation.

Following each quotation, in HTML comments, are the page numbers where that text is found in "The Art of Code." The dissertation is available from <http://repository.upenn.edu/dissertations/AAI3072974/> Despite the "Scholarly Commons" banner which appears at the top of that page, the document is not in the commons; it is for sale. "ScholarlyCommons@Penn also offers links to electronic dissertations of Penn students provided by ProQuest's Digital Dissertations service, though these dissertations are not part of the repository itself. All users have free access to 24-page previews of these dissertations."

### The Art of Code

Arguing that software's increasing abstraction from hardware has defined computer programming practices for the last half-century, this dissertation shows how that abstraction has shaped the aesthetics, politics, and professional culture of programming. Specifically, the dissertation examines how some programmers have adopted a literary approach to coding, describing carefully crafted code as "beautiful," "elegant," "expressive," and "poetic"; writing and reading programs as literary texts; and even producing hybrid artifacts that are at once poems and programs. This project has two central goals: first, to show how identifiably linguistic sensibilities have influenced programming theory and culture; second, to show how programming theory, as a body of knowledge that thinks deeply about the semantics and organization of textual structures, can contribute to the project of literary study.

Finding a literary-like programming practice to be espoused by and reflected in *The Matrix*, Black considers how writerly programming traditions began in the context of early compilers and progressed through the Open Source movement. He argues that "the aesthetics of code itself constituted a powerful motivation" for the sharing of source code - an assertion that rings true to me, since my embarrassment about functional but poorly-written code of mine has certainly made me more reticent to share it. Black makes it clear that he is seeking to characterize existing programming practices in a useful way by referring to literary practices, not trying to "impose a trendy postmodern paradigm ... nor ... define a new 'cyberaesthetic' for the digital era." While the dissertation's project is to place programming within the tradition of literary study, it also aims to renew literary study's ability to understand programming. Black finds that debates about digital culture have "failed to import into literary studies a solid understanding of computing or programming history," and he argues that "to understand the progressive technical and cultural abstractions that have been built atop computing hardware and software, one must also understand the underlying mechanisms of those electronic and coded systems and appreciate the technical and cultural histories underpinning their construction." Black, intending his dissertation to be polemical, moves through programming history until, at the end of the dissertation, he uses object-oriented programming practice to understand James Joyce's writing. He also considers how, while literary studies has sought non-traditional, transgressive topics and approaches, computer programming has at the same time adopted a very traditional concept of literary practice and a desire for classical elegance.

#### 1. Crossed Wires: Historiographies of Code

In the first chapter, Black notes how software has been largely left out of the history of computing, except when particular corporations or programming languages are traced through time. (I would have added to this list of exceptions a small number of high-profile systems such as Doug Engelbart's NLS, also even in that case,

people are more apt to remember the mouse - a hardware innovation - than the software advances that were made.) The chapter's story focuses on how software has been written and shared since the 1940s, arguing that "we need a new way to think computing history within literary studies, one that refuses to treat the computer or computing code as a blank slate on which to build utopian fantasies about identity, embodiment, and textuality, one that instead situates itself in a truly interdisciplinary and informed way among the various fields and philosophies that computing integrates." Describing Turkle's characterization of the PC as modernist and the Macintosh (which keeps users on the image-filled surface) as postmodernist - a division much like Umberto Eco's famous Protestant/Catholic one - Black goes on to quite correctly point out that the MS-DOS command line interface "is itself a 'scintillating surface,' the result of three decades' worth of accumulated abstraction from the real 'guts' of computing hardware." Finding existing literary and cultural theory inadequate, Black provides a new model of "layered abstraction" for the historical study of computing.

In his historical discussion, Black is careful to consider the ENIAC in the context of the times. He declares it a phenomenally fast computer, noting that it was 1000 times as fast as electromechanical IBM Aiken Mark I which preceded it by less than three years. He also notes that, despite Turing's existing model of a simple, general-purpose computer, it was not obvious at first how real digital computers, such as the hardwired ENIAC, strung with patch cables, could be general-purpose. What was needed was the concept of the stored program, first described in the von Neumann architecture - a framework developed in collaboration with Eckert, Mauchly, and others. Black notes that while high-level programming languages were first being developed, a parallel innovation also fostered the development of modern programming -- the sharing of code:

Grace Murray Hopper, one of the Harvard Mark I's first programmers, recalls the informal process for obtaining shared code: "If I needed a sine subroutine, I'd whistle at [a fellow programmer] and say 'Can I have your sine subroutine?'," she remembers, "and I'd copy it out of his notebook."

Black describes how handwritten notebooks of checked code led to the subroutine, and how the stored program and high-level languages allowed software to be commodified. He chronicles the rise of IBM as a business that charged only for hardware, not at all software (interesting, in light of Big Blue's Linux investments today) and the emergence of software-sharing hobbyist home computer programmers alongside non-sharers, such as Microsoft. More recent opposition to software "blackboxing" is also covered, including that by Linus Torvalds, Eric Raymond, and Richard Stallman. He also asserts:

It will be my contention that the humanities has figured "the computer" in a way that has done more to sustain a debilitating divide between the two fields than to integrate them in mutually enabling, creative ways. I will argue here that far from generating useful knowledges or deep understanding of computing within the humanities, "cybertheory" or "cyberstudies" - as the academy calls its theoretical approach to computer-related topics - produces a fantasy about computers that is largely divorced from the culture, history, and technology of computing itself.

The dismantling of the cyberacademic approach begins at The First Conference on Cyberspace in May 1990 at UT Austin, describing the utopian science-fictional approach to computing that began there as "a sort of armchair intergalactic adventure" with "virtually no pretention to comprehend or care about the actual workings of technology." Black writes that "cybertheory cast itself as the last, best hope of cultural studies." He adds:

Since its inception, the goal of cyberstudies has remained constant: to develop an interpretive mode unfettered by reality, to create a space within criticism for futuristic imagining, to perfect the project of cultural studies by producing an interpretive analogue of science fiction. The subject of that fiction is the computer. The plot is the story of how the computer engineers a new global future. ... It is crucial to see how deeply readings of cyberspace, whether optimistic or pessimistic, depend on an unexamined and largely uninformed idea of the computer. ... Whether prophesying doom and gloom or predicting endless utopian liberations, cybertheorists have one thing in common: they use the computer to advance a preconceived political agenda, and they license themselves to do so by strategically ignoring both how computers work and what animates computing culture.

Black goes on to dissect a section ("The Frightening Computer") of Deborah Lupton's 1995 essay, "The Embodied Computer/User," showing how Lupton creates some sort of way to critically discuss the computer even while dismissing the computer's workings as impossible to understand. Black describes how Mark Hansen's writing in *Embodying Technesis* opposes such an idea but ends up privileging the physical layer of computer systems. Black goes on to slam the cultural and historical disconnection of the work in *The Cybercultures Reader* and the conflation of Victorian technologies with modern ones in Herbert Sussman's "Machine Dreams: The Culture of Technology."

As much as I agree with Black when he points out the lack of awareness of technological fundamentals and the divorce of this approach from culture and history, and although I can't find fault with any specific arguments, I still wouldn't come down quite so harshly on Michael Benedikt, Marcos Novak, Donna Haraway, and the rest of the cybertheory crowd. Their academic approaches are more prophetic and poetic, if perhaps less thoroughly grounded, and they are at least connected to some interesting currents of thought in science fiction, utopian writing, and even other fields, such as architecture. Cybertheory can certainly mean something, even if it leaves open a need for well-founded work that is aware of the nature of the computer, computing practices, and technological history. It just won't answer (or even ask) all the questions that humanists should have about computers. Black is probably right, too, about how the obsessive reference to the computer as a vague metaphor clouds our ability to understand it as a part of our technological environment and our actual culture.

Black turns for the beginnings of a technically informed theory to programmers themselves, the "accidental revolutionaries" who have had to articulate their practices, principles, and motivations.

Cyberstudies does not devote much space to the questions that preoccupy hackers. For the hacker, the central issues have to do with where code comes from, who owns it, who writes it, and how good it is. For the cybertheorist, there is just the impersonal, artifactual monolith: the computer ... the cybertheorist's desire to make pronouncements about the nature of computers without actually studying computers results in all manner of buggy arguments.

Black goes on to read "The Cyborg Manifesto" against "The GNU Manifesto." (Hey, Noah and I thought of that - at least, we thought to juxtapose them in *The New Media Reader*.) "Stallman is building a system; Haraway is trying to dismantle one." "Haraway's equation of coding with domination is based on utter ignorance of the 'oppression' that coding itself was undergoing at the very moment she was writing." His conclusion is that "The Cyborg Manifesto" misses the real nature of coding at the moment when it was written:

Indeed, by assuming so blithely that code belongs to the victors, Haraway manages to reinforce the very equations she claims she wants to dismantle. Participating in the sort of logic that Stallman is trying to resist, i.e., the belief that code is automatically the sole, proprietary possession of the dominant economic few, Haraway's manifesto is manifestly misguided."

## 2. At the Edge of Language: The Art of Code

The chapter opens with Eavan Boland's poem "Code," an ode to Grace Murray Hopper. Black then traces the idea of programming as an aesthetic practice, starting with Ada Lovelace. According to Black she is, whatever her actual role in writing programs for Babbage's never-built Analytical Engine, nevertheless a "technological Byronic hero" for being the first to conceive of programming as an art form, and thus for contributing to the figural history of programming. Black traces the history of this concept through Joseph Weizenbaum and Fred Brooks. The next section deals with intellectual property law, noting how the unlikely case *Diamond v. Diehr* (1981) about a method for curing synthetic rubber ended up allowing software patents. Black continues to consider the Unix operating system code as a text that was studied and read closely; he particularly discusses John Lions's 1977 *Commentary on the Unix Operating System*. Black continues with a discussion of Knuth's clear view of writing programs as an art, one that can be "literate" and be like writing poems. The final section is on Perl poetry, which Black traces back to the Oulipo and Noël Arnaud's book *Poèmes algol*, which was written using only the keywords from Algol 60 and inspired by a similar short poem by François Le Lionnais. He quotes the first Perl poem by Larry Wall, a haiku from March 1990, and also prints in its entirety Sharon Hopkins' famous Perl poem "listen." While an interesting beginning to the discussion of Perl poetry, I missed any discussion of the tension between a Perl poem's human-readable and machine-executable nature, of how it is

*hard* to write a Perl poem and where compromises occur in them. Aren't there places where a Perl program could be a better program, or be a better poem? How do such texts negotiate their harsh constraints?

### 3. "Harmonic Concenser Enginium": Object-Oriented Joyce

Black argues here that object-oriented programming (which he describes in some detail, defining abstraction, encapsulation, inheritance, and polymorphism with examples) is a useful concept for understanding *Finnegans Wake*. The chapters discuss the ideas and work of Jean Piaget, Seymour Papert, and Alan Kay. Making this correspondence is not as silly as it sounds, but the metaphor does not seem to be completely anchored. Joyce's writing practice may have been similar to object-oriented programming methodology in some ways, but I missed what the text of *Finnegans Wake* corresponds to. Object-oriented source code itself, with the reader the reader the machine it runs on? Or is the novel's text output from code that is implicit somewhere? Or a binary or bytecode (or syllablecode) form of the source, which was manifested partially in Joyce's notes and letters? Some interesting comments of Black's and some notes about predecessors of his method follow:

Two things are noteworthy about Derrida's application of computing metaphors to Joyce's prose. First, we see how Derrida imports the engineering imagery that for Joyce was profoundly productive and positive, then warps that imagery, by way of the computer, to turn Joyce into an authoritarian sadist. Secondly, Derrida's analogy rests less on any substantive knowledge about programming or computing history than an assumption that programming is necessarily a dictatorial, authoritarian act, one designed to limit, contain, and close off the play of meaning. The assumption is a false one, and as such the analogy is, too. An informed analogy between Joyce and programming theory yields a very different reading of the "programmatic" nature of Joyce's prose, suggesting less that it is a means of controlling the play of meaning than of organizing words so as to maximize their semiotic potential.

In her 1989 book *Writing Joyce: A Semiotics of the Joyce System*, Lorraine Weir suggests that "the operations of John McCarthy's system, 'LISP,' provide another way of configuring that fugal arborescence of textual paradigms [that is] a principal structuring mode in *Ulysses* and *Finnegans Wake*" (94).

Donald Theall has noted Joyce's interest in the mathematical definition of types and their relation to the sigla: "Joyce read Russell's *Introduction to Mathematical Philosophy* with a particular interest in the chapter on the theory of types.... [He] used the formality of mathematical logic in the construction of relationships between sigla... He apparently sensed the problems of 'strange loops' and the infinite regression of classes, hence also the concept of 'meta-levels.'" ... (*James Joyce's Techno-Politics* 167)."

... both Joyce and object-oriented programmers solved the problem of complexity in the same way for the same reasons: both Joyce and the creators of object-oriented programs objected to reductive approaches to complex systems, preferring instead to honor complexity by organizing it; both devised simple means of harnessing the energy complexity generates; both, too, used those means to amplify complexity's power.

### Conclusion: Poetics of *Techne*

A discussion of Heidegger's rift between *techne* and *poiesis* is the background for presenting computer programmers as "presencing forth" *poiesis* within *techne*. Black writes, "As Heidegger understood *poiesis* and *techne* to be inseparable aspects of a larger poetic whole, so I have understood computing history as a deep intertwining of the poetic and technological."

Hacker culture has been explored historically by Stephen Levy, anthropologically by Eric Raymond, philosophically by Pekka Himanen, but it has never been studied aesthetically, as an artistic culture in and of itself. As the first attempt to develop a sustained, serious analysis of computing culture as distinct from literary culture, *The Art of Code* seeks to lay the foundation for an aesthetics of technology that would not accept the Heideggerian split of *techne* from *poiesis*.

