# Code: Notes on an expressive, linguistic medium

Adam Florin 05/19/2011

As popular apprehension to computers and programming wanes, many of the grand, science-fiction notions of what role computing will play in our lives have subsided. A view of code as an expressive medium that enables the thinking of its author has breached the walls of the old "hacker" counter-culture and is rapidly becoming as ordinary as expressing oneself through writing or other creative means. Still, we tend to lavishly endow computers and their processes with a panoply of metaphors, even in the terms coders use to describe their own products: "tools", "libraries", "code", "programs", "applications", "systems", "machines"... How do we as a culture read these artifacts? As textual documents, as designed objects, as engineered mechanisms, as cryptography? I set out to write this paper to clarify my own thinking, sifting through half a century of theory and making note of what I see as the major contemporary trends in critical thinking on the subject: computing is a medium; code is a language; and, as human artifacts, they all reflect back to us symbolic images of our understanding of the world around us.

* * *

Coming around to viewing computing as a medium—"a technology for transforming and transferring messages"[1], akin to print or television—was a fairly lengthy process, and popular opinion only really accepted it once computer had become a mainstream product, available to a mass audience. For the first decades of its existence, computing

[1] From Bolter and Gromala; see next footnote.

was the exclusive domain of academic, scientific research, and as such was conceived of in highly theoretical terms, even endowed with a certain magic that may make current popular attitudes toward computing seem awfully mundane. Jay David Bolter and Diane Gromala dissect the two main trends in early attitudes toward computing, referring to them as "myths of disembodiment".

The first "myth of disembodiment", reigning from computers' very inception through the late '70s, was Artificial Intelligence, a topic pioneered by the eccentric early computer theorist and mathematician Alan Turing:

> During World War II, Turing had helped to build and use the mechanical Bombe and the electronic Colossus to decrypt messages from the Germans' Enigma coding machines, so he knew that computer could function as a technology for transforming and transferring messages—that is, as a medium. But he went in another direction, appropriate for an introverted genius, and became convinced that the digital computer was not just a medium but a mind. For Turing and others who followed him, the computer should not just be a channel for human messages; it should be a thinking machine, capable of producing its own messages.[2]

In this view, computers are so novel, so radically different from earlier technologies, that I think there was a temptation to view them as a sort of alien life force. There's a somewhat cold and spooky tone here, and indeed, most portrayals of AI from that era of popular culture were not benign; consider HAL from *2001: A Space Odyssey*. In fact, the fearful and apprehensive attitude to computers in sci-fi from the period feels a bit like the apes' reaction to the black monolith in that film. This new object was the ultimate "black box": opaque, impenetrable, and therefore—possibly smarter than we are, even God-like.

There's also much to suggest that humanity has always sought to make the unknowable understandable by ascribing humanlike quantities to it. I'm thinking of the origin of myth, polytheistic religions, ghosts—explanation by way of personification. I'm hardly an expert in this subject. Sherry Turkle better explains why computers may have been endowed with psychological interpretations than earlier technologies:

---

[2] Jay David Bolter and Diane Gromala, *Windows and Mirrors*, p. 16

> If you open an computer or a computer toy, you see no gears that turn, no levers that move, no tubes that glow... Even with considerable sophistication, the workings of the computer present no easy analogies with objects or processes that came before, except for analogies with people and their mental processes... The physical opacity of this machine encourages it to be talked about and thought about in psychological terms.[3]

Even today, techno-utopians like mad inventor Ray Kurzweil (author of books with titles such as *The Age of Spiritual Machines*) tell us that computer intelligence is still set to surpass human intelligence. (In fact, he has set a year for it: 2042, conveniently likely to be just within his own lifetime.) While Kurzweil has no doubt provided real insights in his field, the debate has largely cooled down, as many other scholars have moved on. To contemporary readers, there's a troubling logical assumption, made quite easily half a century ago, behind the equivalence of human and machine intelligence. The thinking then was: all human intelligence is abstract symbol manipulation; computers can do abstract symbol manipulation; therefore, computers can achieve or surpass human intelligence. But the very notion that human intelligence consists only of abstract symbol manipulation is an absurdity to us now, an odd relic from the height of mid-century structuralist thought. Second-wave feminism and poststructuralism have encouraged us to celebrate our subjectivity, and contemporary scientific research and pop psychology from authors like Malcolm Gladwell (*Blink*) reassures us that our emotional intelligence is central to our reasoning, and hardly an extraneous artifact of evolution.[4]

The second "myth of disembodiment", per Bolter and Gromala, was "cyberspace", which dominated through much of the '80s and '90s. In this conception, made popular by now-defunct technologies such as "Virtual Reality", computing took on another mystical form, this time as a "space" for escape from physical reality. Again, we can

---

[3] Sherry Turkle, *The Second Self*, p. 22

[4] For an excellent philosophical debunking of AI from a former practitioner of it, I highly recommend Terry Winograd and Fernando Flores' *Understanding Computers and Cognition*, which rests heavily on Heidegger.

look to the sci-fi of the era for clues: in *Tron* or *Neuromancer* (where the term "cyberspace" was coined), the computer is a realm of pure thought which humans enter, leaving their earthly bodies behind—an image reinforced in more recent years by *The Matrix* and *Avatar*. From the advent of the Apple through the early days of the Web (before business interests were able to make use of computing as a commercial medium), computers and the internet still had something of a counter-cultural air, especially to many prominent thinkers in the Bay Area and elsewhere. No doubt the internet gives users opportunity to reflect upon themselves, but in the era of Facebook transparency, we don't ascribe quite as much importance to the fantastical significance of the avatar as then. Based on my own upbringing, I associate this era with Burning Man and rave culture, where people assume party names and separate identities, and thrill at the opportunity of self-reinvention as a counter-cultural rite.

I've been drawing from science fiction references because I see both of these earlier notions of computing as overtly literary in their conception—casting literary identities onto the medium, rather than recognizing it as its own sort of literary medium. In this way, they undervalue computing as a medium by burdening it with fantastical dimensions that made it difficult for the mainstream to penetrate.

* * *

Viewed as a medium, computing is more self-evidently a human construct, rather than any kind of other-worldly intrusion into humanity. By the same token, we're able to view computer programming languages as the human languages they are. "The first and most obvious point is that whenever someone writes a program, it is a program *about* something"[5], write Winograd and Flores (my emphasis). This view of

---

[5] Terry Winograd and Fernando Flores, *Understanding Computers and Cognition* (1987), p. 84

programming as a reflective, authorial act is echoed by Sherry Turkle: "A computer program is a reflection of its programmer's mind."[6] The impetus and inspiration for all computer programs comes from a human desire to represent and model some idea or behavior, often drawn from the physical world. Programmers frequently speak of the "satisfaction" of just seeing an idea or concept in motion[7]. Programming is a profoundly creative act. Matthew Kirschenbaum, an English professor versed in code, describes "programming as *world-making*" (my emphasis again), and views computers as "engines for creating powerful and persuasive models of the world around us". In this way, he links the creative act of programming to fiction (another kind of "world-making"), and laments that his undergrad computer science classes failed to impart to him

> why and how such an activity was connected to the long traditions of humanistic thought I encountered in the classes devoted to my major, reading Leibnitz for example or (better) Jane Austen, surely one of our ultimate system builders and world-makers.[8]

Programmers have increasingly come to celebrate their position as linguistic practitioners above all, inviting numerous parallels to authorship in "natural languages" (as opposed to computer programming languages), in the form of designations such as "code poet" (which, it must be said, is sometimes used pejoratively). Kirschenbaum cites Donald E. Knuth, a computer scientist who has placed a large emphasis on code legibility, explaining that while code must of course be interpreted by the machine, *human-readability* must be a primary goal, increasing the code's value as a document or artifact of human thought or understanding. To this end, he championed what he called "literate programming", calling programming "a very

---

[6] Sherry Turkle, *The Second Self* (1984), p. 19

[7] This sentiment comes up frequently in the TV documentary *Hackers* (1984) about early Macintosh programmers, directed and produced by my father, Fabrice Florin.

[8] Matthew Kirschenbaum, "Hello Worlds", *The Chronicle of Higher Education* (2009)

personal activity", and declaring that "the practitioner of literate programming can be regarded as an essayist, whose main concern is with exposition and excellence of style."[9]

The emphasis on the power of programming languages as models for human thought and action points to a greater understanding of the power of language at large. Winograd and Flores point the field of "speech act theory, as originated by the philosopher J. L. Austin (*How to Do Things with Words*, 1962)"[10]. Language *is* action, they tell us:

> Language is a form of human social action...The shift from language as description to language as action is the basis of speech act theory, which emphasizes the *act* of language rather than its representational role...
> Speech acts create commitment... To be human is to be the kind of being that generates commitments, through speaking and listening.[11]

Thus, computer programming may be considered a *speech act*: a linguistic instruction which sets up a commitment to be executed by the computer (which of course is in no way supposed to be considered an autonomous agent or anything human-like here—but is rather an implement of the speech act).

"It is important to keep in mind that computer code, and computer programs, are not machine creations and machines talking to themselves, but writings by humans", Florian Cramer informs us[12], in an essay about the liminal space between programming and natural languages, proposing "a digital poetry which reflects the intrinsic textuality of the computer" and hoping that "computers and digital poetry might teach us to pay more attention to codes and control structures coded into all language". He points to earlier "poetics of formal instruction", such as Fluxus pieces, and points out that

---

[9] Donald E. Knuth, "Literate Programming" (1984)

[10] Winograd and Flores, p. 58

[11] Winograd and Flores, p. 76

[12] Florian Cramer, "Digital Code and Literary Text" (2011)

“Western music is an outstanding example of an art which relies upon written formal instruction code”:

> Self-reflexive injokes such as "B-A-C-H" in Johann Sebastian Bach's music, the visual figurations in the score of Erik Satie's "Sports et divertissements" and finally the experimental score drawings of John Cage shows that, beyond a merely serving the artwork, formal instruction code has an aesthetic dimension and intellectual complexity of its own. In many works, musical composers have shifted instruction code from classical score notation to natural human language.

The comparison of computer source code to a musical score, and computer execution to musical performance, recurs elsewhere. It’s a paradigm which ended up neatly fitting the notions of language and interpretation described by philosopher Nelson Goodman in his work *Languages of Art* (1968). John Lee applies this thinking to code, describing a computer program as “a highly notational text that forms the basis of some kind of implementation”, and finally noting that “the position of the composer, then, does seem to closely parallel that of the computer programmer. A notational work is produced, which the computer implements.”[13]

Early computer science furnished the basic concepts from discrete mathematics (functions, variables) that were necessary to create a meaningful symbolic processing language. In the last few decades, as technology improvements have enabled most programmers to keep a safe distance from the physics and details of hardware-level implementation (writing in “machine language”, the numeric sequences of digits which undergird all computer use), programming languages for the last few decades have taken a more linguistic turn, becoming playgrounds for experimentation in formal symbolic languages for human thought. In fact, these languages, as tools of abstract thought[14], may be one of computers’ greatest contributions to human thought in and of themselves. Today, coders who are so inclined may invent their own programming languages in the abstract, based on their own intuitions, and implement them on top of

---

[13] John Lee, "Goodman's Aesthetics and the Languages of Computing", *Aesthetic Computing* (2006)

[14] Howard Rheingold titled one book *Tools for Thought*.

older, more "low-level" languages. In just such a way, Yukihiro Matsumoto designed the Ruby programming language based on the "principle of least surprise" (a dubious claim perhaps—least surprising to whom?), and implemented it on top of C, a more canonical and lower-level language. He describes his motivation, clearly if somewhat indelicately:

> Often people, especially computer engineers, focus on the machines. They think, "By doing this, the machine will run faster. By doing this, the machine will run more effectively. By doing this, the machine will something something something." They are focusing on machines. But in fact we need to focus on humans, on how humans care about doing programming or operating the application of the machines. We are the masters. They are the slaves. [15]

This has been the general trend in the last few decades of computing: "bring the computer to the problem, not the problem to the computer". Architect Malcolm McCullough writes, "the history of programming may be understood as largely a matter of increasing abstraction"[16]. Computer programming is steadily becoming further integrated into popular culture, as the low-level programming is managed by technical specialists, and floods of novice programmers and part-time geeks gain access to the power of world-making on the computer. As such, we see programs more and more as documents of human thought, artifacts of human endeavor—and as such, are able to read them more and more as *texts* within the greater medium of computing.

* * *

In the past decade, the culture of code is increasingly becoming a household craft, neither specifically radical nor corporate, but more just as an everyday language that can express whatever the programmer desires. The earliest programming languages and concepts, such as object-oriented programming, may have seemed predominantly

---

[15] Bill Venners, "The Philosophy of Ruby: A Conversation with Yukihiro Matsumoto, Part I" (2003)

[16] Malcolm McCullough, *Abstracting Craft* (1996), p. 97

modeled on mid-century corporate hierarchy and organizational theory by writers such as Herbert Simon; such a reading can easily be found in the language of seminal books like the *Design Patterns*[17] book by a group of computer scientists colloquially referred to as "the Gang of Four", in which structures bear names such as "worker", "factory", "delegate", "command", etc. But today, languages and the communities around them, while still employing some of this basic structural thinking—can be more easily seen through the lens of lifestyle; Ruby coders bundle reusable components into "gems", which often have slickly-designed web presences and strong branding, even for relatively small bits of code. The Ruby community is particularly concerned with appearances, and I must confess that when browsing for gems, I will privilege ones that have a visual language close to my own design values before I assess the code itself. The new open source movement is largely taking place on a site called GitHub (tagline: "social coding") where open-source projects are easily "forked" so that each coder can make their own custom modifications, with a chance that the original author will "pull" those latter changes into the original codebase. What began as an idealistic counter-culture now looks more like sharing ideas on Twitter, or photos on Facebook. The latest trend is to continue to pull code further from its origins as machine language, toward natural language. Gems such as Cucumber for Ruby enable users to write "business-readable" text (where "business" means "layperson") for use in verifying the proper functionality of an existing codebase.

In the era of Web 2.0, when visual design, transparency, and social networks became the cornerstones of the web and computing as a medium in general, code has become more available to the greater humanities-minded public. Digital Humanities and Library Sciences programs have been teaching basic data-mining code to their students. Coders such as Adrian Holovaty have been advocating for a new model of

---

[17] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, *Design Patterns* (1994)

"programmer as journalist", a position taken up by Cindy Royal[18]. New search engine tools such as Wolfram Alpha make even search more closely resemble coding, and even the somewhat curmudgeonly Don Norman, an outspoken proponent of wordless clarity in design, suggested that search engines "are becoming 'answer services' controlled through their command line interfaces"[19]. In the last decade, teenagers seeking to customize their MySpace pages wrote and exchanged CSS "codes" (and by pluralizing the word, somehow restored some of the cryptographic connotations therein!).

And while the word "interactive"—with its uncomfortable implication that the computer has some kind of proper agency—remains commonly in use, products such as the RjDj app for iOS have begun to use the word "reactive" to describe the user experience. This may in fact be more descriptive. If computing is a medium, then it must be passive by default—and if programs are all human-authored texts, then can it be said that computers ever really initiate an interaction? This thinking may just be a useful shorthand, as we don't give as much credence to computers as we once did: we don't, for example, construe an error message as the computer "yelling" at us[20].

Reactive computing can also be seen through the metaphor of the mirror, which I found prevalent in writing on user interface. Once we've conceived of programming languages as systems of thought, and created worlds within them, what do we get back? The "techno-determinist" camp (Kittler et al.) suggests that we are deeply affected. Other authors suggest that we receive an image of ourselves. In the context of his interactive media art, David Rokeby writes: "Mirrors give us back an image with which to identify. We look at the marks we have made on our world to give us a sense

---

[18] Cindy Royal, "The Journalist as Programmer: A Case Study of *The New York Times* Interactive News Technology Department", *International Symposium in Online Journalism The University of Texas at Austin* (2010)

[19] Donald A. Norman, "UI Breakthrough-Command Line Interfaces", *Interactions* (2007)

[20] However, this may be different for children, as Deborah, one adolescent example given by Sherry Turkle in *The Second Self*, would begin to cry whenever she saw an error message for this reason. (p. 144)

of our significance"[21]. The mirror metaphor seems especially prevalent among those close to camera-based computer vision art, in which case the computer is really re-presenting an image captured from the physical world, much as a mirror does. Rokeby refers to his *Very Nervous System* piece, and Bolter and Gromala lean on Daniel Rozin's excellent *Wooden Mirror* piece, in which the visitor's image is recreated on physical wooden "pixels". For these authors (Rokeby included), the importance is that the mirror not only draw attention to us, but to itself, as well:

> In the hands of technologists, a medium evolves towards apparent transparency... The message (as per McLuhan) that such a medium conveys may be powerful, but it is generally unintentional. Of course, interactive artists intentionally express themselves through the opacities and idiosyncrasies of the media that they create. These media reflect, but also guide and transform, the gestures of the interactor. [22]

This is the topic Bolter and Gromala expressly set out to prove (unfortunately never citing Rokeby!): the basic underlying belief among Don Norman and the interaction design set (whom they label as the "structuralists") is that user interface should always strive for "transparency"—which, to them, is another myth. They advocate making better use of "the rhythms of transparency and reflectivity" (where "reflectivity" may be understood as self-reflectivity, or Rokeby's opacity), pointing to the Macintosh interface as a positive example.[23]

Other writers are drawn to the mirror example for other reasons, citing the important of Lacan's "mirror phase" in child development. Sherry Turkle notes that "some children become far more explicit ... about seeing the computer as a mirror of the mind. These are children who make explicit use of computational metaphors to think

---

[21] David Rokeby, "Transforming Mirrors: Subjectivity and Control in Interactive Media", *Critical Issues in Electronic Media* (1995), p. 153

[22] Rokeby, p. 144

[23] Bolter and Gromala, p. 74

about themselves."[24] In yet other examples, authors question the nature of Narcissus' fixation on the image in the pool (as McLuhan did).

"Computers are a metatechnology, almost infinitely flexible and bristling with potential," Rokeby continues. Indeed, computing, and especially computer programming, remains hard to pin down because it is so general. Don Norman proposed specialized "information appliances"[25], and indeed, as we see in the huge app market for iOS, and the approval from prominent authors[26], that such specialization in our applications is beginning to take place. But this is just the case for "apps" with complete user interfaces; and even as "domain-specific languages" gain traction, coders will always need to have a general-purpose programming language under their belt—just as it would be improbable for a child in the US to learn legalese but not English.

I like to joke that Computer Science is the study of how the universe would look had it been designed by humans—overloaded with numerous, conflicting metaphors, emerging from the accidental intersection of high-minded theory and impetuous action. My mentor for this paper, Sara Roberts, pointed out to me that philosophy is often driven by the technology of the day; my favorite example is that of Freud characterizing psychosexual energies such as repression in the mechanical terms of the steam engine. Code, however, offers the opportunity to materially participate in technology while gaining new, rich, symbolic modes of thinking for self-reflexive thought. A large part of the act of programming is the thinking that precedes it, and the way that thinking is shaped through expression and articulation in code of the world as it is, and the behaviors of objects within it. I look forward to seeing the wider adoption of coding practice by an ever-growing humanities community, employing this new expressive medium to make sense of the world through writing and doing.

---

[24] Turkle, p. 155

[25] Donald A. Norman, *The Invisible Computer* (1998)

[26] Chris Anderson, "The Web is Dead. Long Live the Internet", *Wired* (2010)