X. PURPOSE

Program or Be Programmed

Digital technology is programmed. This makes it biased toward those with the capacity to write the code. In a digital age, we must learn how to make the software, or risk becoming the software. It is not too difficult or too late to learn the code behind the things we use—or at least to understand that there is *code behind their interfaces. Otherwise, we are at the mercy of those who do the programming, the people paying them, or even the technology itself.*

One of the US Air Force generals charged with building and protecting the Global Information Grid has a problem: recruitment. As the man in charge of many of the Air Force's coolest computer toys, he has no problem attracting kids who want to fly drones, shoot lasers from satellites, or steer missiles into Persian Gulf terrorist camps from the safety of Shreveport. They're lining up for those assignments. No, the general's challenge is finding kids capable of *programming* these weapons systems—or even having the education, inclination, and mental discipline required to begin learning programming from scratch.

Raised on commercial video games that were themselves originally based on combat simulation technologies, these recruits have enviable reflexes and hand-eye coordination. They are terrific virtual pilots. Problem is, without an influx of new programmers capable of maintaining the code and fixing bugs—much less upgrading and innovating new technologies—the general cannot keep his operation at mission readiness. His last resort has been to give lectures at education conferences in which he pleads with high schools to put programming into their curriculums.

That's right: America, the country that once put men on the moon, is now falling behind most developed and many developing nations in computer education. We do not teach programming in most public schools. Instead of teaching programming, most schools with computer literacy curriculums teach *programs*. Kids learn how to use popular spreadsheet, word processing, and browsing software so that they can operate effectively in the high-tech workplace. These basic skills may make them more employable for the entry-level cubicle jobs of today, but they will not help them adapt to the technologies of tomorrow.

Their bigger problem is that their entire orientation to computing will be from the perspective of users. When a kid is taught a piece of software as a subject, she'll tend to think of it like any other thing she has to learn. Success means learning how to behave in the way the program needs her to. Digital technology becomes the immutable thing, while the student is the movable part, conforming to the needs of the program in order to get a good grade on the test.

Meanwhile, kids in other countries—from China to Iran—aren't wasting their time learning how to use off-the-shelf commercial software packages. They are finding out how computers work. They learn computer languages, they write software and, yes, some of them are even taught the cryptography and other skills they need to breach Western cyber-security measures. According to the Air Force general, it's just a matter of a generation before they've surpassed us.

While military superiority may not be everyone's foremost goal, it can serve as a good indicator of our general competitiveness culturally and economically with the rest of the world. As we lose the ability to program the world's computers, we lose the world's computing business as well. This may not be a big deal to high-tech conglomerates who can as easily source their programming from New Delhi as New Hampshire. But it should be a big deal to us.

Instead, we see actual coding as some boring chore, a working-class skill like bricklaying, which may as well be outsourced to some poor nation while our kids play and even design video games. We look at developing the plots and characters for a game as the interesting part, and the programming as the rote task better offloaded to people somewhere else. We lose sight of the fact that the programming—the code itself—is the place from which the most significant innovations emerge.

Okay, you say, so why don't we just make sure there are a few students interested in this highly specialized area of coding so that we can keep up militarily and economically with everyone else? Just because a few of us need to know how to program, surely that doesn't mean we *all* need to know programming, does it? We all know how to drive our cars, yet few of us know how our automobiles actually work, right?

True enough, but look where that's gotten us: We spend an hour or two of what used to be free time operating a dangerous two-ton machine and, on average, a full workday each week paying to own and maintain it.[9] Throughout the

9. The Bureau of Labor Statistics (http://www.bls.gov/) updates these figures yearly.

twentieth century, we remained blissfully ignorant of the real biases of automotive transportation. We approached our cars as consumers, through ads, rather than as engineers or, better, civic planners. We gladly surrendered our public streetcars to private automobiles, unaware of the real expenses involved. We surrendered our highway policy to a former General Motors chief, who became secretary of defense primarily for the purpose of making public roads suitable for private cars and spending public money on a highway system. We surrendered city and town life for the commuting suburbs, unaware that the bias of the automobile was to separate home from work. As a result, we couldn't see that our national landscape was being altered to manufacture dependence on the automobile. We also missed the possibility that these vehicles could make the earth's atmosphere unfit for human life, or that we would one day be fighting wars primarily to maintain the flow of oil required to keep them running.

So considering the biases of a technology before and during its implementation may not be so trivial after all. In the case of digital technology, it is even more important than usual. The automobile determined a whole lot about how we'd get from place to place, as well as how we would reorganize our physical environment to promote its use. Digital technology doesn't merely convey our bodies, but ourselves. Our screens are the windows through which we are experiencing, organizing, and interpreting the world in which we live. They are also the interfaces through which we express who we are and what we believe to everyone else. They are fast becoming the boundaries of our perceptual and conceptual apparatus; the edge between our nervous systems and everyone else's, our understanding of the world and the world itself.

If we don't know how they work, we have no way of knowing what is really out there. We cannot truly communicate, because we have no idea how the media we are using bias the messages we are sending and receiving. Our senses and our thoughts are already clouded by our own misperceptions, prejudices, and confusion. Our digital tools add yet another layer of bias on top of that. But if we don't know what their intended and accidental biases are, we don't stand a chance of becoming coherent participants in the digital age. **Programming is the sweet spot, the high leverage point in a digital society. If we don't learn to program, we risk being programmed ourselves**.

The irony here is that computers are frightfully easy to learn. Programming is immensely powerful, but it is really no big deal to learn. Back in the 1970s, when computers were supposedly harder to use, there was no difference between operating a computer and programming one. Better public schools offered computer classes starting in the sixth or seventh grade, usually as an elective in the math department. Those of us lucky to grow up during that short window of opportunity learned to think of computers as "anything machines." They were blank slates, into which we wrote our own software. The applications we wrote were crude and often rather pointless—like teaching the computer to list prime numbers, draw pictures with text, or, as in my own final project, decide how to prioritize the decisions of an elevator car.

I'm sure only one or two of us actually graduated to become professional programmers, but that wasn't the point. All of us came to understand what programming is, how programmers make decisions, and how those decisions influence the ways the software and its users function.
For us, as the mystery of computers became the science of programming, many other mysteries seemed to vanish as well. For the person who understands code, the whole world reveals itself as a series of decisions made by planners and designers for how the rest of us should live. Not just computers, but everything from the way streets are organized in a town to the way election rules (are tilted for a purpose vote for any three candidates) begin to look like what they are: sets of rules developed to promote certain outcomes. Once the biases become apparent, anything becomes possible. The world and its many arbitrary systems can be hacked.

Early computers were built by hackers, whose own biases ended up being embedded in their technologies. Computers naturally encouraged a hacker's approach to media and technology. They made people less interested in buying media and a bit more interested in making and breaking it. They also turned people's attention away from sponsored shows and toward communicating and sharing with one another. The problem was that all this communicating and sharing was bad for business.

So the people investing in software and hardware development sought to discourage this hacker's bias by making interfaces more complex. The idea was to turn the highly transparent medium of computing into a more opaque one, like television. Interfaces got thicker and more supposedly "user friendly" while the real workings of the machine got buried further in the background. The easy command-line interface (where you just type a word telling the machine what you want it to do) was replaced with clicking and dragging and pointing and watching. It's no coincidence that installing a program in Windows required us to summon "The Wizard"—not the helper, the puppy, or even that "Paper Clip Man." No, we needed the Wizard to re-mystify the simple task of dragging an application into the applications folder, and maybe a database file somewhere else. If we had been privy to everything the Wizard was doing on our behalf, then we may have even been able to uninstall the entire program without purchasing one of those hard drive sweeping utilities. Instead, we were told not to look behind the curtain.

It was all supposedly safer that way. Accepting the computer salesman's pitch as technological truth, we bought the false premise that the more open a device was to us, the more open it was to every bad person out there. Better to buy a locked-down and locked-up device, and then just trust the company we bought it from to take care of us. Like it used to say on the back of the TV set: *Hazard of electric shock. No user serviceable parts inside*. Computing and programming were to be entrusted to professionals. Consumers can decorate their desktops the way they like, and pick which programs to purchase, but heaven forbid they trust an unauthorized vendor or, worse, try to do something themselves. They must do everything through the centralized applications program, through the exclusive carrier, and not try to alter any of it. The accepted logic is that these closed technologies and systems are safer and more dependable.

Of course none of this is really true. And the only way you'd really know this is if you understood programming. Fully open and customizable operating systems, like Linux, are much more secure than closed ones such as Microsoft Windows. In fact, the back doors that commercial operating systems leave for potential vendors and consumer research have made them more vulnerable to attack than their open source counterparts. This threat is compounded by the way commercial vendors keep their source code a secret. We aren't even to know the ways we are vulnerable. We are but to trust. Even the Pentagon is discouraged from developing its own security protocols through the Linux platform, by a Congress heavily lobbied to promote Windows.[10]

Like the military, we are to think of our technologies in terms of the applications they offer right out of the box instead of how we might change them or write our own. We learn what our computers already do instead of what we can make them do. This isn't even the way a kid naturally approaches a video game. Sure, a child may play the video game as it's supposed to be played for a few dozen or hundred hours. When he gets stuck, what does he do? He goes online to find the "cheat codes" for the game. Now, with infinite ammunition or extra-strength armor, he can get through the entire game. Is he still playing the game? Yes, but from outside the confines of the original rules. He's gone from player to cheater.

After that, if he really likes the game, he goes back online to find the modification kit—a simple set of tools that lets a more advanced user change the way the game looks and feels. So instead of running around in a dungeon fighting monsters, a kid might make a version of the game where players run around in a high school fighting their teachers—much to the chagrin of parents and educators everywhere. He uploads his version of the game to the Internet, and watches with pride as dozens or even hundreds of other kids download and play his game, and then comment about it on gamers' bulletin boards.

10. See Richard Clarke, *Cyberwar: The Next Threat to National Security* (New York: HarperCollins, 2010).

The more open it is to modification, the more consistent software becomes with the social bias of digital media.

Finally, if the version of the game that kid has developed is popular and interesting enough, he just may get a call from a gaming company looking for new programmers. Then, instead of just creating his own components for some other programmer's game engine, he will be ready to build his own.

These stages of development—from player to cheater to modder to programmer—mirror our own developing relationship to media through the ages. In preliterate civilizations, people attempted to live their lives and appease their gods with no real sense of the rules. They just did what they could, sacrificing animals and even children along the way to appease the gods they didn't understand. The invention of text gave them a set of rules to follow—or not. Now, everyone was a cheater to some extent, at least in that they had the choice of whether to go by the law, or to evade it. With the printing press came writing. The Bible was no longer set in stone, but something to be changed—or at least reinterpreted. Martin Luther posted his ninety-five theses, the first great "mod" of Catholicism, and later, nations rewrote their histories by launching their revolutions.

Finally, the invention of digital technology gives us the ability to program: to create self-sustaining information systems, or virtual life. These are technologies that carry on long after we've created them, making future decisions without us. The digital age includes robotics, genetics, nanotechnology, and computer programs—each capable of self-regulation, self-improvement, and self-perpetuation. They can alter themselves, create new versions of themselves, and even collaborate with others. They grow. These are not just things you make and use. These are emergent forms that are biased toward their own survival. Programming in a digital age means determining the codes and rules through which our many technologies will build the future—or at least how they will start out.

The problem, as I explained in the introduction, is that we haven't actually seized the capability of each great media age. We have remained one dimensional leap behind the technology on offer. Before text, only the Pharaoh could hear the words of the gods. After text, the people could gather in the town square and hear the word of God read to them by a rabbi. But only the rabbi could read the scroll. The people remained one stage behind their elite. After the printing press a great many people learned to read, but only an elite with access to the presses had the ability to write. People didn't become authors; they became the gaming equivalent of the "cheaters" who could now read the Bible for themselves and choose which laws to follow.

Finally, we have the tools to program. Yet we are content to seize only the capability of the last great media renaissance, that of writing. We feel proud to build a web page or finish our profile on a social networking site, as if this means we are now full-fledged participants in the cyber era. We remain unaware of the biases of the programs in which we are participating, as well as the ways they circumscribe our newfound authorship within their predetermined agendas. Yes, it is a leap forward, at least in the sense that we are now capable of some active participation, but we may as well be sending text messages to the producers of a TV talent show, telling them which of their ten contestants we think sings the best. Such are the limits of our interactivity when the ways in which we are allowed to interact have been programmed for us in advance.

Our enthusiasm for digital technology about which we have little understanding and over which we have little control leads us not toward greater agency, but toward less. We end up at the mercy of voting machines with "black box" technologies known only to their programmers, whose neutrality we must accept on faith. We become dependent on search engines and smart phones developed by companies we can only hope value our productivity over their bottom lines. We learn to socialize and make friends through interfaces and networks that may be more dedicated to finding a valid advertising model than helping us find one another.

Yet again, we have surrendered the unfolding of a new technological age to a small elite who have seized the capability on offer. But while Renaissance kings maintained their monopoly over the printing presses by force, today's elite is depending on little more than our own disinterest. We are too busy wading through our overflowing inboxes to consider how they got this way, and whether there's a better or less frantic way to stay informed and in touch. We are intimidated by the whole notion of programming, seeing it as a chore for mathematically inclined menials than a language through which we can re-create the world on our own terms.

We're not just building cars or televisions sets—devices that, if we later decide we don't like, we can choose not to use. We're tinkering with the genome, building intelligent machines, and designing nanotechnologies that will continue where we leave off. The biases of the digital age will not just be those of the people who programmed it, but of the programs, machines, and life-forms they have unleashed. In the short term, we are looking at a society increasingly dependent on machines, yet decreasingly capable of making or even using them effectively. Other societies, such as China, where programming is more valued, seem destined to surpass us—unless, of course, the other forms of cultural repression in force there offset their progress as technologists. We shall see. Until push comes to shove and geopolitics force us to program or perish, however, we will likely content ourselves with the phone apps and social networks on offer. We will be driven toward the activities that help distract us from the coming challenges—or stave them off—rather than the ones that encourage us to act upon them.

But futurism is not an exact science, particularly where technology is concerned. In most cases, the real biases of a technology are not even known until that technology has had a chance to exist and replicate for a while. Technologies created for one reason usually end up having a very different use and effect. The "missed call" feature on cell phones ended up being hacked to give us text messaging. Personal computers, once connected to phone lines, ended up becoming more useful as Internet terminals. Our technologies only submit to our own needs and biases once we hack them in one way or another. We are in partnership with our digital tools, teaching them how to survive and spread by showing them how they can serve our own intentions. We do this by accepting our roles as our programs' true users, rather than subordinating ourselves to them and becoming the used.

In the long term, if we take up this challenge, we are looking at nothing less than the conscious, collective intervention of human beings in their own evolution. It's the opportunity of a civilization's lifetime. Shouldn't more of us want to participate actively in this project?

Digital technologies are different. They are not just objects, but systems embedded with purpose. They act with intention. If we don't know how they work, we won't even know what they want. The less involved and aware we are of the way our technologies are programmed and program themselves, the more narrow our choices will become; the less we will be able to envision alternatives to the pathways described by our programs; and the more our lives and experiences will be dictated by their biases.

On the other hand, the more humans become involved in their design, the more humanely inspired these tools will end up behaving. We are developing technologies and networks that have the potential to reshape our economy, our ecology, and our society more profoundly and intentionally than ever before in our collective history. As biologists now understand, our evolution as a species was not a product of random chance, but the forward momentum of matter and life seeking greater organization and awareness. This is not a moment to relinquish our participation in that development, but to step up and bring our own sense of purpose to the table. It is the moment we have been waiting for.

For those who do learn to program see the rest of the world differently as well.

Even if we don't all go out and learn to program—something any high school student can do with a decent paperback on the subject and a couple of weeks of effort—we must at least learn and contend with the essential biases of the technologies we will be living and working with from here on.

I've endeavored to explain ten of the most significant ones here, as well as how to turn them from liabilities into opportunities. But you will surely continue to find others. I encourage you to explore them, come up with your own strategies, and then share them with others—including me.

If living in the digital age teaches us anything, it is that we are all in this together. Perhaps more so than ever.