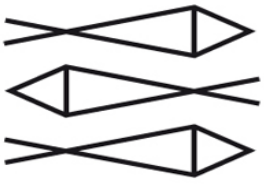
LIFE IN CODE

A PERSONAL HISTORY OF TECHNOLOGY

ELLEN ULLMAN



LIFE IN CODE

A Personal History of Technology

ELLEN ULLMAN

MCD FARRAR, STRAUS AND GIROUX | NEW YORK

Outside of Time

REFLECTIONS ON THE PROGRAMMING LIFE

1994

I.

People imagine that programming is logical, a process like fixing a clock. Nothing could be further from the truth. Programming is more like an illness, a fever, an obsession. It's like riding a train and never being able to get off.

The problem with programming is not that the computer isn't logical—the computer is terribly logical, relentlessly literal-minded. Computers are supposed to be like brains, but in fact they are idiots, because they take everything you say at face value. I can say to a toddler, "Are yew okay tewday?" and the toddler will understand. But it's not possible for a programmer to say anything like that to a computer. The compiler

complains; finds a syntax error; won’t translate your program into the zeros and ones of the machine. It’s not that a program can’t be made to act *as if* it understands—it can. But that’s just a trick, a trick of the code.

When you are writing code, your mind is full of details, millions of bits of knowledge. This knowledge is in human form, which is to say rather chaotic, coming at you from one perspective, then another, then a random thought, then something else important, then the same thing with a what-if attached. For example, try to think of everything you know about something as simple as an invoice. Now try to tell an alien how to prepare one. That is programming.

A computer program is an algorithm that must be written down in order, in a specific syntax, in a strange language that is only partially readable by regular human beings. To program is to translate between the chaos of human life and the line-by-line world of computer language.

You must not lose your own attention. As the human-world knowledge tumbles about in your mind, you must keep typing, typing. You must not be interrupted. Any break in your listening causes you to lose a line here or there. Some bit comes; then—oh no—it’s leaving, please come back. It may not come back. You may lose it. You will create a bug and there’s nothing you can do about it.

Every single computer program has at least one bug. If you are a programmer, it is guaranteed that your work has errors. These errors will be discovered over time, most coming to light after you’ve moved on to a new job. But your name is on the program. The code library software keeps a permanent record card of who did what and when. At the old job, they will say terrible things about you after you’ve gone. This is normal life for a programmer: problems trailing behind you through time, humiliation in absentia.

People imagine that programmers don’t like to talk because they prefer machines to people. This is not completely true. Programmers don’t talk

because they must not be interrupted.

This inability to be interrupted leads to a life that is strangely asynchronous with the one lived by other human beings. It's better to send email than to call a programmer on the phone. It's better to leave a note on the chair than to expect the programmer to come to a meeting. This is because the programmer must work in mind-time but the phone rings in real time. Similarly, meetings are supposed to take place in real time. It's not just ego that prevents programmers from working in groups—it's the synchrony problem. To synchronize with other people (or their representation in telephones, buzzers, and doorbells) can only mean interrupting the thought train. Interruptions mean certain bugs. You must not get off the train.

I used to have dreams in which I was overhearing conversations I had to program. Once, I had to program two people making love. In my dream they sweated and tumbled while I sat with a cramped hand writing code. The couple went from gentle caresses to ever-widening passions, and I despaired as I tried desperately to find a way to express the act of love in the computer language called C.

II.

I once had a job where I didn't talk to anyone for two years. Here was the arrangement: I was the first engineer hired by a startup software company. In exchange for large quantities of stock that might be worth something someday, I was supposed to give up my life.

I sat in a large room with two recently hired engineers and three Sun workstations. The fans of the machines whirred; the keys of the keyboards clicked. Occasionally, one or another of us would grunt or mutter. Otherwise, we did not speak. Now and then, I would have a temper outburst in which I pounded the keyboard with my fists, setting off a barrage of beeps. My colleagues might look up but never said anything about this.

Once a week, I had a five-minute meeting with my boss. I liked him; he was genial; he did not pass on his own anxieties about working in a startup. At this meeting I would routinely tell him I was on schedule. Since being on schedule is a very rare thing in software engineering, he would say, “Good, good, see you next week.”

I remember watching my boss disappear down the row of cubbyhole partitions. He always wore clothes that looked exactly the same: khaki pants and a checked shirt of the same pattern. So, week to week, the image of his disappearing down the row of partitions remained unchanged. The same khaki pants, the same pattern in the checked shirt. “Good, good, see you next week.”

Real time was no longer compelling. Days, weeks, months, and years came and went without much physical change in my surroundings. Surely I was aging. My hair must have grown, I must have cut it; it must have grown again. Gravity must have been working on my sedentary body, but I didn’t notice. I only paid attention to my back and shoulders because they seized up on me from long sitting. Later, after I left the company, there was a masseuse on staff. That way, even the back and shoulders could be soothed —all the better to keep you in your seat.

What was compelling was the software. I was making something out of nothing, I thought, and I admit the software had more life for me than my brief love affair, my friends, my cat, my house, my neighbor who was stabbed and nearly killed by her husband. I was creating (“creating”—that is the word we used) a device-independent interface library. One day, I sat in a room by myself surrounded by computer monitors from various manufacturers. I remember looking at the screens of my companions and saying, “Speak to me.”

I completed the interface library in two years and left the company. On my last day on the job, the financial officer gave me a check: it was a payment to buy back most of my stock. I knew this was coming. When I

was hired I'd signed an agreement: the price of leaving before five years was the return of the stock. Still, I didn't feel free or virtuous. I put the check in my pocket, then got drunk at my farewell party.

Five years later, the company went public. For the engineers who'd stayed, the original arrangement was made good: in exchange for giving up seven years of their lives, they became very, very wealthy. As for me, I bought a car. A red one.

III.

Frank was thinking he had to get closer to the machine. Somehow, he'd floated up. Up from memory heaps and kernels. Up from file systems. Up through utilities. Up to where he was now: an end-user query tool. Next thing, he could find himself working on general ledgers, invoices—God—*financial reports*. Somehow, he had to get closer to the machine.

Frank hated me. Not only was I closer to the machine, I had won the coin toss to get the desk near the window. Frank sat in full view of the hallway, and he was farther from the machine.

Frank was nearly forty. His wife was pregnant. Outside, in the parking lot (which he couldn't see through my window), his new station wagon was heating up in the sun. Soon he'd have a kid, a wife who had just quit her job, a wagon with a child carrier, and an end-user query tool. Somehow, he had to get closer to the machine.

Here are the reasons Frank wanted to be closer to the machine: The machine means midnight dinners of Diet Coke. It means unwashed clothes and bare feet on the desk. It means anxious rides through mind-time that have nothing to do with the clock. To work on things used only by machines or other programmers—that's the key. Programs and machines don't care how you live. They don't care when you live. You can stay, come, go, sleep, or not. At the end of the project looms a deadline, the

terrible place where you must get off the train. But in between, for years at a stretch, you are free: free from the obligations of time.

To express the idea of being “closer to the machine,” an engineer refers to “low-level code.” In regular life, “low” usually signifies something bad. In programming, “low” is good. Low is better.

If the code creates programs that do useful work for regular human beings, it is called “higher.” Higher-level programs are called “applications.” Applications are things that people use. Although it would seem that usefulness by people would be a good thing, from a programmer’s point of view, direct people-use is bad. If regular people, called “users,” can understand the task accomplished by your program, you will be paid less and held in lower esteem. In the regular world, the term “higher” may be better, but in programming higher is worse. High is bad.

If you want money and prestige, you need to write code that only machines or other programmers understand. Such code is “low.” It’s best if you write microcode, a string of zeros and ones that only a processor reads. The next best thing is assembler code, a list of instructions to the processor, but readable if you know what you’re doing. If you can’t write microcode or assembler, you might get away with writing in the C language or C+ +. C and C+ + are really sort of high, but they’re considered “low.” So you still get to be called a “software engineer.” In the grand programmer scheme of things, it’s vastly better to be a “software engineer” than a “programmer.” The difference is thousands of dollars a year and a potential fortune in stock.

My office mate Frank was a man vastly unhappy in his work. He looked over my shoulder, everyone’s shoulder, trying to get away from the indignity of writing a program used by normal human beings. This affected his work. His program was not all it should have been, and for this he was punished. His punishment was to have to talk to regular people.

Frank became a sales-support engineer. Ironically, working in sales and having a share in bonuses, he made more money. But he got no more stock options. And in the eyes of other engineers, Frank was as “high” as one could get. When asked, we said, “Frank is now in sales.” This was equivalent to saying he was dead.

IV.

Real techies don’t worry about forced eugenics. I learned this from a real techie in the cafeteria of a software company.

The project team is having lunch and discussing how long it would take to wipe out a disease inherited recessively on the X chromosome. First come calculations of inheritance probabilities. Given a population of a given size, one of the engineers arrives at a wipe-out date. Immediately another suggests that the date could be moved forward by various manipulations of the inheritance patterns. For example, he says, there could be an education campaign.

The six team members then fall over one another with further suggestions. They start with rewards to discourage carriers from breeding. Immediately they move to fines for those who reproduce the disease. Then they go for what they call “more effective” measures: Jail for breeding. Induced abortion. Forced sterilization.

Now they’re hot. The calculations are flying. Years and years fall from the final doom-date of the disease.

Finally, they get to the ultimate solution. “It’s straightforward,” someone says. “Just kill every carrier.” Everyone responds to this last suggestion with great enthusiasm. One generation and—bang—the disease is gone.

Quietly, I say, “You know, that’s what the Nazis did.”

They all look at me in disgust. It’s the look boys give a girl who has interrupted a burping contest. One says, “This is something my wife would say.”

When he says “wife,” there is no love, warmth, or goodness in it. In this engineer’s mouth, “wife” means wet diapers and dirty dishes. It means someone angry with you for losing track of time and missing dinner. Someone *sentimental*. In his mind (for the moment), “wife” signifies all programming-party-pooping, illogical things in the universe.

Still, I persist. “It started as just an idea for the Nazis, too, you know.”

The engineer makes a reply that sounds like a retch. “This is how I know you’re not a real techie,” he says.

V.

A descendant of Italian princes directs research projects at a well-known manufacturer of UNIX workstations. I’m thrilled. In my then five years of being a consultant, the director is the first person to compliment me on what I am wearing to the interview.

It takes me a while, but I soon see I must forget all the usual associations with either Italians or princes. There will be no lovely long lunches that end with deftly peeled fruit. There will be no well-cut suits of beautiful fabrics. The next time I am wearing anything interesting, the director (I’ll call him Paolo) tells me I look ridiculous.

Paolo’s Italian-ness has been replaced, outer-space-pod-like, with some California New Age, Silicon Valley engineering creature. He eats no fat. He spoons tofu-mélange stuff out of a Tupperware container. Everything he does comes in response to beeps emitted from his UNIX workstation: he eats, goes to meetings, goes rollerblading in the parking lot, buys and sells stock, calls his wife solely in response to signals he has programmed into his calendar system. (The clock on his wall has only the number twelve on it.) Further, Paolo swears he has not had a cold since the day he decided that he would always wear two sweaters. Any day now, I expect to see him get out of his stock-option Porsche draped in garlic.

I know that Paolo has been replaced because I have met his wife. We are at a team beer-fest in the local programmer hangout on a Friday evening. It's full of men in tee shirts and jeans. Paolo's wife and I are the only people wearing makeup. She looks just the way I expect a no-longer-young Italian woman to look—she has taken time with her appearance, she is trying to talk to people. Across the swill of pitchers and chips glopped with cheesy drippings, she eyes me hopefully: another grown-up woman. At one point, she clucks at Paolo, who is loudly describing the effects of a certain burrito. "The only thing on earth that instantly turns a solid into a gas," he says.

The odder Paolo gets, the more he fits in with the research team. One engineer always eats his dessert first (he does this conscientiously; he wants you—dares you—to say something; you simply don't). Another comes to work in something that looks suspiciously like his pajamas. To join this project, he left his wife and kids back east. He obviously views the absence of his family as a kind of license: he has stopped shaving and (one can't help noticing) he has stopped washing on a regular basis. Another research engineer comes to work in shorts in all weather; no one has ever seen his knees covered. Another routinely makes vast changes to his work the day before deadlines; he is completely unmoved by any complaints about this practice. And one team member screens all email through a careful filter, meaning most mail is deposited in a dead-letter file. This last engineer, the only woman permanently on the project, has outdone everyone on oddness: she has an unlisted work phone. To reach her, you must leave a message with her manager. The officially sanctioned asynchrony of the unlisted phone amazes me. I have never seen anything like it.

These research engineers can be as odd as they like because they are very, very close to the machine. At their level, it is an honor to be odd. Strange behavior is expected, it's respected, a sign that you are intelligent and as close to the machine as you can get. Any decent software engineer can have a private office, come and go at all hours, exist out of normal time.

But to be permanently and sincerely eccentric—this is something only a senior research engineer can achieve.

In meetings, they behave like children. They tell each other to shut up. They call each other idiots. They throw balled-up paper. One day, a team member screams at his Korean colleague, “Speak English!” (A moment of silence follows this outburst, at least.) It’s like dropping in at the day-care center by mistake.

They even behave like children when their Japanese sponsors come to visit. The research is being funded through a chain of agencies and bodies that culminates in the Board of Trade of Japan. The head of the sponsoring department comes with his underlings. They all wear blue suits. They sit at the conference table with their hands folded neatly in front of them. When they speak, it is with the utmost discretion; their voices are so soft, we have to lean forward to hear. Meanwhile, the research team behaves badly, bickers, has the audacity to ask when they’ll get paid.

The Japanese don’t seem to mind. On the contrary, they appear delighted. They have received exactly what their money was intended to buy. They have purchased bizarre and brilliant Californians who can behave any way they like. The odd behavior reassures them: Ah! These must be real top-rate engineers!

VI.

We are attending conventions. Here is our itinerary: we will be traveling closer and closer to the machine. Our journey will be like crossing borders formed by mountain ranges. On the other side, people will be very different.

We begin “high,” at a conference of computer trainers and technical writers. Women are everywhere. There is a great deal of nail polish, deep red, and briefcases of excellent leathers. In the cold, conditioned air of the conference hall drifts a faint, sweet cloud of perfume.

Next we travel to Washington, D.C., to an applications development conference, the Federal Systems Office Expo. It is a model of cultural diversity. Men, women, whites, blacks, Asians—all qualified applicants are welcome. Applications development ("high-level," low-status, and relatively low-paying) is the civil service of computing.

Now we move west and lower. We are in California to attend a meeting of SIGGRAPH, the graphics special-interest group of the Association for Computing Machinery (ACM). African Americans have virtually disappeared. Young white men predominate, with many Asians among them. There are still some women. This is the indication—the presence of just a few women—that we are getting ever closer to the heart of the machine.

From here we descend rapidly into the deep, low valleys of programming. We go first to an operating-systems interest group of the ACM. Then, getting ever closer to hardware, we attend a convention of chip designers. Not a female person in clear sight. If you look closely, however, you might see a few young Chinese women sitting alone—quiet, plainly dressed, succeeding at making themselves invisible. For these are gatherings of young men. This is the land of tee shirts and jeans, the country of perpetual graduate-studenthood.

Later, at a software-vendor developers conference, company engineers proudly call themselves "barbarians" (although they are not really as "low" as they think they are). In slides projected onto huge screens, they represent themselves in beards and animal skins, holding spears and clubs. Except for the public-relations women (the scent of Chanel Nº 5 rising from the sidelines), there is only one woman (me).

A senior engineer once asked me why I left full-time engineering for consulting. At the time, I had never really addressed the question, and I was surprised by my own answer. I muttered something about feeling out of

place. “Excuse me,” I found myself saying, “but I’m afraid I find the engineering culture very teen-age boy puerile.”

This engineer was a brilliant man, good-hearted, and unusually literate for a programmer. I had great respect for him, and I really did not mean to offend him. “That’s too bad,” he answered as if he meant it, “because we obviously lose talent that way.”

I felt immense gratitude at this unexpected opening. I opened my mouth to go on, to explore the reasons for the cult of the boy engineer.

But immediately we were interrupted. The company was about to have an interdivisional water-balloon fight. For weeks, the entire organization had been engaged in the design of intricate devices for the delivery of rubberized inflatable containers filled with fluid. Work had all but stopped; all “spare brain cycles” were involved in preparations for war.

The friendly colleague joined the planning with great enthusiasm. The last I saw of him, he was covering a paper napkin with a sketch of a water-balloon catapult.

Here is a suggested letter home from our journey closer to the machine: Software engineering is a meritocracy. Anyone with the talents and abilities can join the club. However, if rollerblading, Frisbee playing, and water-balloon wars are not your idea of fun—if you have friends you would like to see often, children you would like to raise—you are not likely to stay long.

VII.

I once designed a graphical user interface with a man who wouldn’t speak to me. My boss hired this man without letting anyone else sit in on the interview; my boss lived to regret it.

I was asked to brief my new colleague, and, with a third member of the team, we went into a conference room. There we covered two whiteboards

with lines, boxes, circles, and arrows in four marker colors. After about half an hour, I noticed that the new hire had become very agitated.

“Are we going too fast?” I asked him.

“Too much for the first day?” said the third.

“No,” said our new man, “I just can’t do it like this.”

“Do what?” I asked. “Like what?”

His hands were deep in his pockets. He gestured with his elbows. “Like this,” he said.

“You mean design?” I asked.

“You mean in a meeting?” asked the third.

No answer from our new colleague. A shrug. Another elbow gesture.

Something terrible was beginning to occur to me. “You mean talking?” I asked.

“Yeah, talking,” he said. “I can’t do it by talking.”

By this time in my career, I had met many strange engineers. But here was the first one who wouldn’t talk at all. Besides, this incident took place before the existence of standard user interfaces like Windows and Motif, so we had a lot of design work to do. Not talking was certainly going to make things difficult.

“So how can you do it?” I asked.

“Mail,” he said immediately, “send me email.”

So, given no choice, we designed a graphical user interface by email.

Corporations across North America and Europe are still using a system designed by three people who sent email, one of whom barely spoke at all.

VIII.

Pretty graphical interfaces are commonly called “user friendly.” But they are not really your friends. Underlying every user-friendly interface is a terrific human contempt.

The basic idea of a graphical interface is that it does not allow anything alarming to happen. You can pound on the mouse button all you want, and the system should prevent you from doing anything stupid. A monkey can pound on the keyboard, your cat can run across it, your baby can bang it with a fist, but the system should not crash.

To build such a crash-resistant system, the designer must be able to imagine—and disallow—the dumbest action. He or she cannot simply rely on the user's intelligence: who knows who will be on the other side of the program? Besides, the user's intelligence is not quantifiable; it's not programmable; it cannot protect the system. The real task is to forget about the intelligent person on the other side and think of every single stupid thing anyone might possibly do.

In the designer's mind, gradually, over months and years, there is created a vision of the user as imbecile. The imbecile vision is mandatory. No good, crash-resistant system can be built except if it's done for an idiot. The prettier the user interface, and the fewer odd replies the system allows you to make, the dumber you once appeared in the mind of the designer.

The designer's contempt for your intelligence is mostly hidden deep in the code. But, now and then, the disdain surfaces. Here's a small example: You're trying to do something simple, like back up files on your Mac. The program proceeds for a while, then encounters an error. Your disk is defective, says a message, and below the message is a single button. You absolutely must click this button. If you don't click it, the program hangs there indefinitely. So—your disk is defective, your files may be bolloxed up, and the designer leaves you only one possible reply: You must say, "OK."

IX.

The computer is about to enter our lives like blood into the capillaries. Soon, everywhere we look, we will see pretty, idiot-proof interfaces

designed to make us say, “OK.”

A vast delivery system for retail computing is about to come into being. The system goes by the name “interactivity.” The very word—interactivity—implies something good and wonderful. Surely a response, a reply, an answer is a positive thing. Certainly it signifies an advance over something else, something bad, something that doesn’t respond, reply, or answer. There is only one problem: what we will be interacting with is a machine.

Interactive services are supposed to be delivered “on demand.” What an aura of power—demand! See a movie, order seats to a basketball game, make hotel reservations, send a card to Mother—all services waiting for us on our telephones, televisions, computers. Midnight, dawn, or day. Sleep or order a pizza: it no longer matters exactly what we do when. We don’t need to involve anyone else in the satisfaction of our needs. We don’t even have to talk. We get our services when we want them, free from the obligations of regularly scheduled time. We can all live closer to the machine.

“Interactivity” is misnamed. It should be called “asynchrony”: the engineering culture coming to everyday life.

In the workplace, home office, sales floor, we will be “talking” to programs that are beginning to look surprisingly alike: all full of animated little pictures we are supposed to pick, like push buttons on a toddler’s toy. The toy is supposed to please us. Somehow, it is supposed to replace the satisfactions of transacting meaning with a mature human being, in the confusion of a natural language, together, in a room, at a touching distance.

As the computer’s pretty, helpfully waiting face (and contemptuous underlying code) penetrates deeply into daily life, the cult of the boy engineer comes with it. The engineer’s assumptions and presumptions are in the code. That’s the purpose of the program, after all: to sum up the intelligence and intentions of all the engineers who worked on the system over time, tens and hundreds of people who have learned an odd and highly specific way of doing things. The system contains them. It reproduces and

re-enacts life as engineers know it. Soon we may all be living the programming life: alone, floating in mind-time, disdainful of anyone far from the machine.