Pass the Torch: Enabling the Next Generation of Hackers

by Tom Pittman

Hacking, though intrinsically value-free, involves an ethical dimension. A good hack expresses creativity, simplicity, and purpose; a bad hack may be unoriginal or corrupt the system. The main difference between a good hack and an excellent commercial product lies in its size and price; it is quantitative, not qualitative. Furthermore, although hacking is essentially a solo activity, it has important social consequences. Nobody takes up the hacker's hatchet unaided. In some cases it is another hacker who personally inspires the young initiate; in other cases it is access to appropriate tools and good sample code. Either way, ultimately it was a person who did it. Hacking is a valuable asset to society. We repay our obligation to our mentors by inspiring the next generation of hackers through good hacks and personal encouragement.

I used to think of programming as "beating back the advance of entropy" by injecting information and order into an otherwise entropic system. Then I discovered a different sense in which the complexity I added was itself an expression of entropy: as we pour energy and information into the technology that drives modern society, it thereby becomes more chaotic and less predictable.

As I write this, I am involved in the software redesign of a product, the original requirements of which proved so complex that the lead programmer had to eliminate half the features just to get it working. Most of those feature requirements were added back in when I took it on, and I also find the complexity overwhelming. It is interesting to observe that the same creeping feature creature that bit off Jim's head is now poised, jaws agape, over mine — never mind that I have a lot more experience and professional skills than he ever did.

On my first Mac, after upgrading it to 512K, I ran a lean 70K System file on a RAM disk and experienced an amazing leap in productivity over all previous computer systems in my career, large or small. Today I do even better with a rather corpulent 600K System file (not counting fonts), but I dread the day I must switch to a 1.6+ megabyte System 7. "Pink," the developing system software Apple contributed to its IBM alliance, reportedly has already over a million lines of code and 10,000 objects.

My first public software product was a Tiny Basic interpreter that fit in a 2K ROM, and I wrote it in less than a month. Now I sell a 600K HyperTalk compiler that with far better tools took some three years to get right.

The first Apple computer was built in a garage from a handful of logic gates on a 10" board back when most microprocessors sold for $400. Today's high-end Apple computer is still built on about the same size board with the same number of chips (and the CPU still costs something like $400), but we are talking about large-scale custom integrated circuits surface-mounted on four-layer boards with wiring traces so tiny they cannot be done by hand.

My father's first car, a Ford Model A, was simple enough in construction that he could do all his own maintenance with $20 worth of tools. I similarly maintained my first vehicle with tools that I already had. The car I bought this year, however, is so complex that the shop manual alone costs over $40; a computer many times the price of the car is required for a simple tune-up.

Part of my college tuition was paid from photography profits. I was no Ansel Adams, but I had a well-equipped darkroom capable of good color wedding photographs. Today all that is replaced in the industry by high-tech color print machines and 24-bit color editing software on Macintosh computers too expensive for my budget.

Everywhere you look, western culture is replacing simple methods with complex ones, manual efforts with computer technology. And everywhere the sophistication required to master this technology is zooming out of sight. Who is going to do it?

In America, the world leader in technological innovation for 100 years, advance comes as a curious mixture of big business professionalism and amateur tinkering ("hacking"). The aircraft industry was started as a hobby in a bicycle shop, but now it is the exclusive domain of a half-dozen huge aerospace companies. Computers were invented in research labs funded by large government grants, but they did not begin to remake modern technology until microprocessor-based computer systems became affordable to hackers. The spreadsheet, probably the single most significant software innovation of all time, was invented by a student on an Apple II. A culture of computer phreaks growing up on personal computers carried innovative ideas into industry,

where it's hard to find an electronic gadget any more that is not controlled by a microprocessor.

Software engineering technology and new programming tools are increasing by leaps and bounds, but the required complexity in new computer-based products goes up exponentially. Frederick Brooks wrote in *IEEE Computer* magazine a few years back that there are no silver bullets to solve the software crisis. What he meant is that we cannot expect any magical cures to the problem. And he is right. Engineering in general, and software engineer-ing in particular, can only extend known technology in predictable ways. But the personal computer industry was built by hackers, not engineers. Only science fiction writers — and readers — dared to predict what has come about in two decades. To the rest of the world it was a big surprise, the magical silver bullet. True magic is not predictable.

There is a fundamental difference between engin-eering and hacking. An engineer tries to reduce risks, to build on and improve existing technology to accom-plish known goals in controlled steps. The hacker, like the proverbial fool jumping in where angels fear to tread, ignoring probable failure and potential consequences, optimistically sets out to do the im-possible — and often succeeds. The engineer is like the expert skiers I watched in the winter Olympics a few years back: in full control of every muscle, every turn perfectly executed. The hacker is like the grace-less fellow who stormed down the hill, skipping and sliding on every turn: you were sure he would spill out like so many before him who looked equally out of control. This guy somehow did not take a fall on that particular trip down the slope — and he won the gold medal. The engineer is a professional paid to carry the technology forward in small incremental steps; the hacker works for love on his own resources and has the freedom to create true innovation. The engineer is trained; the hacker is inspired.

How, then, do we inspire the hackers and breed technological winners? If we knew that, we could apply sound engineering techniques to develop them, and they would be engineers, not hackers. A hacker is an artist, and like the artists squandering controversial government grants in the news today, most of their output is worthless. But you cannot know *a priori* which hackers are going to produce and which will fail. Indeed, most will fail to deliver any positive benefit to society. Some may even fall into destructive potholes producing obscenity and computer viruses. But we still encourage artists in general, and we should still encourage hacking.

The best inspiration to excellence and artistry is excellence and artistry. The best inspiration to excellent hacking is likewise excellent hacking. In the "good old days" when professions were, for lack of understanding, more art than technology, the next generation of professionals were trained by a process known as apprenticeship. Today we call the same process mentoring, but it is somewhat less formal. Successful older hackers should seek out and encourage promising young hackers to help them develop the attitudes and skills that will carry the tradition of innovation and excellence forward. Young, ambitious comers could do worse than to listen once in a while to the old fogies. Obviously this works better in a stable craft like stained glass or music, but there is room for mentoring in our profession also. Michael Hogan, my supervisor and mentor at the first full-time job I held, instilled in me his (and now my) philosophy of programming.

Another way for the excellence to pass on to the next generation is by means of inspired and inspiring program code. I learned the Macintosh mostly by reading other people's code through a MacsBug disassembly window. Bill Atkinson and Andy Hertzfeld and the whole Mac team have been an inspiration to us by virtue of the excellent software they released to the world.

As important as mentoring and inspiration-by-example may appear to us, however, all of the people I asked indicated that the dominant enabling influence for them was the availability of adequate tools. The two most important tools are, of course, a computer and a programming language. It almost does not matter which computer or what language, as long as they are available and offer opportunity to do "wonderful" things. "Available" in this context means that they are essentially free, that is, (at least initially) subsidized by other purposes. The MIT hackers in Steven Levy's book used an academic computer that was free after hours. Apple Computer got its start because MOS Technology came out with a $20 microprocessor when the average CPU chip sold for $300. A year earlier MITS made a similar splash by selling a whole computer (kit) for $400, about the same price as the CPU in it sold from Intel.

Peter Nagel tells me, "I had never written a line of code of any sort (except possibly an Excel macro and small Red Ryder scripts) before I got my hands on HyperCard shortly after its original release. At that time, I was between jobs and sat down to write a stack to handle timekeeping and billing chores that I still use in my law practice. Within several months, I bought a copy of Lightspeed Pascal, read its manual and a couple of other books on Pascal, and began writing short little XCMDs to improve my billing stacks." Now he is selling MasterScript, a serious commercial programming tool, to other programmers. An earlier version of this same software substantially influenced some of the features added to HyperCard 2.0. Peter concludes, "The

result of all this is that I am deeply indebted to HyperCard for having introduced me to fundamental programming concepts and techniques."

HyperCard turns out to have a disproportionate influence in opening up computer programming to people. Swami Gurupremananda, another nascent hacker not quite so far along, even went so far as to buy his first Macintosh computer just so he could run HyperCard. "With my increasing knowledge of HyperCard," he said, "I, who have not taken even one course in programming, started making stacks to run our correspondence course. Now all our work is done in HyperCard. The IBMs were forced into early retirement!" Most people, however, are more like Susan Leschr, who told me, "The fact that HyperCard came with my computer (1987) had a lot to do with it." Brian Molyneaux, the CompileIt product manager at Heizer Software, reports that a number of our customers use CompileIt as a stepping stone along the path from HyperCard to conven-tional programming languages such as Pascal and C.

Bill Atkinson reportedly offered HyperCard to Apple on the condition that it ship free in every Macintosh. That is the availability that develops unforeseen hackers and inspires new programmers. Present Apple/Claris policy appears to be based on seeing the vast majority of Mac owners who have not yet discovered HyperCard — or simply lack the hacker mentality — and a few enlightened companies using it in-house for prototyping and other "throw-away" programs. This ignores the fact that HyperCard performs a valuable social service if and only if it is available (read "free" again) to a large number of people, most of whom will never take advantage of it. Bill had the right vision.

Bill Atkinson and Andy Hertzfeld have gone on to other ventures, which I assume and hope will benefit us as much as HyperCard and the Mac did. We must not, however, abdicate our own responsibility to benefit society and leave it all to such luminaries. Peter Nagel is producing software tools to help other programmers along the path he has come. Swami Gurupremananda gets online on CompuServe and elsewhere, answering questions from beginners not far behind himself. Each of us can do our part to help new programmers with advice and tools.

Trying to analyze hackers and the hacker mentality tends to leave us with more questions that it answers. But unanswered questions need not stand between us and the next generation of hackers. Indeed, questions and puzzles often motivate and energize hackers. Maybe we can do some hacking on the questions raised here.

I conclude this essay with a challenge: How were you helped to become what you are? What have you done to pass that help along and repay your debt to society? Our only hope of rescue from exponential complexity is likely to come from some new hack not yet thought of. Perhaps you will invent it; perhaps you can only help some other hacker along the way, and he will invent it. Who can know? Quoting the medieval monastic Bernard, Isaac Newton said, "If I have seen farther than others, it is because I stood on the shoulders of giants." You could be the giant under the feet of the next generation of Newtons.