CONTROVERSY

United We Fall, or Killing the Goose that Laid the Golden Egg

Steve Johnson
Melismatic Software

From time to time it is useful to step back from the day to day pressures and try to get some vision of the larger trends in the industry. A particularly good time to do this is when changing jobs, in a (sometimes vain) attempt to avoid becoming a big fish in a dry pond.

One of the biggest changes in the computer industry over the last twenty years has been the growth of application portability. Twenty years ago, most computing was done on mainframes, and applications that ran on those mainframes had rental fees sufficent to buy one of today's workstations each month! The assumption was that people were willing to pay from 20% to 50% of the yearly hardware rental to get the application software and associated tools. This led to some very crude line-oriented text editors being rented for thousands of dollars a month, for example.

At the same time, manufacturers produced systems so idiosyncratic that even standard languages could not always be run compatibly (I remember with particular fondness that the IBM mainframe system would crash if you had more than 50 comment cards at the start of your Fortran program—small is beautiful!).

Well, we software folks changed all that. The invention of highly portable languages, and, with Unix, highly portable operating systems, meant that we could move applications from larger to smaller (and cheaper) machines, or, conversely, write applications on small machines and run them on big ones. At the same time, with the advent of minicomputers, microcomputers, and supercomputers, we started seeing computer prices (which used to lie in a rather narrow range of

$1-10 million) expand outwards to over five orders of magnitude ($5 thousand to $50 million). The traditional pricing algorithms in the industry were put under great stress by this situation. In the old days, you charged $50 thousand for your application, and that was the end of it; a few percent of the annual rental of the machine. If you try that with a $5 thousand machine, you get $125.

But, suppose you could port that $125 low-end program and run it on the high-end machine. You have just saved $49,875. What a business! Needless to say, the computer manufacturers and software vendors didn't like the sound of money leaking out of their cash registers. In the old days, the application would be written in assembler and depend so heavily on the manufacturer's operating system that there was no hope of moving it; in our brave new world, both the OS and language are portable, so the barriers are a lot lower.

The first thing the manufacturers did to prevent the erosion of their income was to distribute in binary only. This re-established the traditional tie between the application and the hardware, and put a finger in the hole in the fiscal dike. There has been a slow and steady rise of emulators, both hardware and software, that have eaten away at these market barriers, but by and large this strategy has been successful.

Competition is usually good for the consumers, who find that in heavily competitive environments the price drops. And changing from source to binary distribution only postponed the problem of competition, because now the real competition was coming from other hardware boxes. As systems and languages became more portable, and as standards groups became more powerful, machine cycles became more and more interchangable; instead of Apples and Oranges, we had aspirin and aspirin.

Now, economists tell us that when two things look identical to a consumer, the consumer will want to pay the same price for them. The way to get a higher price for something is first to make it look different, then make it look better. This is where marketing comes in; the point of slogans such as "System V: consider it standard" and "The network is the computer" is to persuade you ultimately to pay more for System V cycles or for Sun cycles than you might pay for OSF cycles or HP cycles. This is not to say that there are not real differences in these products that in fact make them worth more to some people than to others. And there is a thin line between pointing out legitimate differences between products (a service to customers) and puffing and hyping a product (a disservice to customers). But on the whole, when one manufacturer has a good idea today, it isn't long before the others have a pretty good copy.

The other effect rocking the industry is the incredible decrease in the cost of computer cycles. Things can get cheaper without becoming a commodity, or (more rarely) vice versa, but cycles have done both, to a degree that is almost without parallel in history. In 1910, a Steinway grand piano cost about the same as a luxury automobile. In 1990, a Steinway grand piano still cost about the same as a luxury automobile. In 1960, a medium-sized computer cost about a hundred Steinways. In 1990, a computer of equivalent power cost a hundredth of a Steinway. If house prices had dropped as much over the same period, you could buy your parents' four bedroom ranch house today for about the cost of a Big Mac, fries, and a shake. While the underlying hardware technology has provided for most of the cost drop, the role of portable software in being able to exploit new hardware quickly should not be minimized. It we hadn't built it, they wouldn't have come.

The whole point of this excursion into the past is to try to understand the future. And every trend suggests that by the year 2000, across a very large set of customers, the cost of software will dominate the hardware cost of computer systems. More strikingly, it is already true, and will become more true, that the cost of using and maintaining a system will dominate the cost of buying it; that is, the salaries paid to the people using the hardware will dominate the hardware costs. In fact, in many situations the improvement in user productivity in moving from old fashioned character oriented interfaces to modern mouse/menu/icon interfaces can pay the total cost of the new hardware.

On the face of it, this is a wonderful situation for software. A few years back, people would routinely pay, say, $6 thousand for a PC system, and their productivity improvement was so great that tens of millions of people bought them. This investment was roughly 75% hardware, and 25% software cost. In a few years, the same, or even more powerful, hardware will cost $500. Since the job done by the PC is still worth roughly $6 thousand to the customer, it would seem that the software should be able to charge several thousand at least, give the customer more power for one third the money, and the software vendors can at least retain, and possibly increase their revenues.

Now, this is not what most customers believe will happen. Most expect that the software packages that cost $1500 a few years ago will soon be available for a hundred or two hundred dollars. In fact, some early signs of this are starting to appear. It sure puzzles me why this should be the case. Hardware and software are very different disciplines, with very different tools and development cycles. If hardware costs drop an order of magnitude, there is no more a priori reason to think that the price of software should drop than to think that the price of a Steinway should drop.

Now, historically software has been both overpriced and underpriced. It has been overpriced because the cost of manufacturing the thousandth or millionth copy of a piece of software is only a few percent of the price to the end user. It has been underpriced because historically (especially with system software) the development costs have been so high that these costs have been difficult to recover in pure software pricing alone. Typically this software was sold bundled, and some of the profit on the hardware was used to fund the system software development.

In this situation, system software becomes a natural monopoly. The first successful product in the field sells enough at a high price to recover the development cost, and then the price drops quickly down to an amount that is only slightly higher than the cost of manufacture. The second company in this field must sell near this low cost, and so cannot make enough money to recover their development costs. If they have any sense, they don't try. If there are several people in this market they compete like mad with each other, the price to the consumer is very low, and the companies in the field have trouble remaining healthy. All this tends to encourage a single supplier in each market who gets filthy rich and still makes it difficult for anyone else to enter the market and make any money at all. Microsoft has this kind of edge in several markets, and has done very well for itself; a few years back, IBM made itself top dog with a similar strategy.

So, what is a rational software supplier to do? In this situation, you win by being first into a market, and, lacking that, to make people think you have a better product (in effect, move the market over to your product, so you can be first again). At the very least, you should try to be different enough that at least some people see you as having the product of choice.

What have the computer makers done? They have joined organizations like OSF and UNIX International that seem dedicated to reducing the differences between their members' products! The smaller the difference between the products, the more directly they compete, the lower the prices, and the sicker the vendors become. Joining together preserves the illusion of control, important in a battered industry (it appears that people would rather have their hands on the wheel and drive off the cliff than be thrown over naked). In the short term, it is good for the customer, since software prices drop to irrationally low levels. In the longer term, the natural monopoly that results is bad for the customer, since the customer pays higher prices than if there were competition, and is likely to find slower and less responsive technical development as well.

To recapitulate this argument, we software folks have developed and refined a portability technology that has made computer cycles into a commodity, made software applications into a natural monopoly, is clobbering a lot of computer makers, and will soon drive a lot of software makers out of business, making a lot of us lose our jobs. We have killed the goose that laid the golden egg. Gloomy, no?

I think a lot of software people will experience some wrenching changes between now and the year 2000, but I also think it will be very possible to have a successful software career in the next decade. First and foremost, keep your eyes open. Things are changing, and you will have to change too. Stay flexible and willing to learn. Keep up with the latest technology. Another tip is to look for companies in a niche—stock trading, gene splicing, etc. Both hardware and software companies exploiting vigorous niches have done well right through the current recession. Another tip is to insulate yourself from hardware; the hardware industry is crashing and burning badly, and it is far from clear who the survivors will be. Maybe you want to be on the sidelines for the titanic battles ahead.

Also, notice that the independent software vendors have been much less enthusiastic about OSF and UI than the hardware vendors.

The most successful such companies recognize that, while they need to be aware of the industry trends and to be able to play in that game, their real future lies in the ways in which they are different from, and better than, their peers. As programmers, we would do well to remember that as well.

[submitted Nov. 25, 1992; revised Jan. 4, 1993; accepted Jan. 14, 1993]