Long Live Closed-Source Software!

There's a reason the iPhone doesn't come with Linux.

by Jaron Lanier

If you've just been cornered by Martha Stewart at an interdisciplinary science conference and chastised for being a wimp, you could only be at one event: Sci Foo, an experimental, invitation-only, wikilike annual conference that takes place at Google headquarters in Mountain View, California. There is almost no preplanned agenda. Instead, there's a moment early on when the crowd of scientists rushes up to blank poster-size calendars and scrawls on them to reserve rooms and times for talks on whatever topic comes to mind. For instance, physicist Lee Smolin, sci-fi author Neal Stephenson, and I talked about the relationship between time and math (touching on ideas presented in my October 2006 column).

The wimp comment was directed at me, and Martha was right. I hadn't stood up for myself in a group interaction. I've always been the shy one in the schoolyard. Back in the 1980s, I was drawn to the possibility that virtual reality would help extend the magical, creative qualities of childhood into adulthood. Indeed, the effect of digital technology on culture has been exactly that, but childhood is not entirely easy. If Lee hadn't forged through the crowd to create our session, I never would have done it. What made Martha's critique particularly memorable, though, is that her observation was directly relevant to what emerged from Sci Foo as the big idea about the future of science.

It wasn't official, of course, but the big idea kept popping up: Science as a whole should consider adopting the ideals of "Web 2.0," becoming more like the community process behind Wikipedia or the open-source operating system Linux. And that goes double for synthetic biology, the current buzzword for a superambitious type of biotechnology that draws on the techniques of computer science. There were more sessions devoted to ideas along these lines than to any other topic, and the presenters of those sessions tended to be the younger ones, indicating that the notion is ascendant.

It's a trend that seems ill-founded to me, and to explain why, I'll tell a story from my early twenties. Visualize, if you will, the most transcendentally messy, hirsute, and otherwise eccentric pair of young nerds on the planet. One was me; the other was Richard Stallman. Richard was distraught to the point of tears. He had poured his energies into a celebrated project to build a radically new kind of computer called the LISP Machine. It wasn't just a regular computer running LISP, a programming language beloved by artificial intelligence researchers. Instead it was a machine patterned on LISP from the bottom up, making a radical statement about what computing could be like at every level, from the underlying architecture to the user interface. For a brief period, every hot computer-science department had to own some of these refrigerator-size gadgets.

It came to pass that a company called Symbolics became the sole seller of LISP machines. Richard realized that a whole experimental subculture of computer science risked being dragged into the toilet if anything happened to that little company—and of course everything bad happened to it in short order.

So Richard hatched a plan. Never again would computer code, and the culture that grew up with it, be trapped inside a wall of commerce and legality. He would instigate a free version of an ascendant, if rather dull, program: the Unix operating system. That simple act would blast apart the idea that lawyers and companies could control software culture. Eventually a kid named Linus Torvalds followed in Richard's footsteps and did something related, but using the popular Intel chips instead. His effort yielded Linux, the basis for a vastly expanded open-software movement.

But back to that dingy bachelor pad near MIT. When Richard told me his plan, I was intrigued but sad. I thought that code was important in more ways than politics can ever be. If politically correct code was going to amount to endless replays of dull stuff like Unix instead of bold projects like the LISP Machine, what was the point? Would mere humans have enough energy to carry both kinds of idealism?

Twenty-five years later, that concern seems to have been justified. Open wisdom-of-crowds software movements have become influential, but they haven't promoted the kind of radical creativity I love most in computer science. If anything, they've been hindrances. Some of the youngest, brightest minds have been trapped in a 1970s intellectual framework because they are hypnotized into accepting old software designs as if they were facts of nature. Linux is a superbly polished copy of an antique, shinier than the original, perhaps, but still defined by it.
Before you write me that angry e-mail, please know I'm not anti–open source. I frequently argue for it in various specific projects. But a politically correct dogma holds that open source is automatically the best path to creativity and innovation, and that claim is not borne out by the facts.

Why are so many of the more sophisticated examples of code in the online world—like the page-rank algorithms in the top search engines or like Adobe's Flash—the results of proprietary development? Why did the adored iPhone come out of what many regard as the most closed, tyrannically managed software-development shop on Earth? An honest empiricist must conclude that while the open approach has been able to create lovely, polished copies, it hasn't been so good at creating notable originals. Even though the open-source movement has a stinging countercultural rhetoric, it has in practice been a conservative force.

> Why did the adored iPhone come out of what many regard as the most closed software-development shop on Earth?

There were plenty of calls at Sci Foo for developing synthetic biology along open-source lines. Under such a scheme, DNA sequences might float around from garage experimenter to garage experimenter via the Internet, following the trajectories of pirated music downloads and being recombined in endless ways.
A quintessential example of the open ideal showed up in Freeman Dyson's otherwise wonderful piece about the future of synthetic biology in a recent issue of *The New York Review of Books*. MIT bioengineer Drew Endy, one of the enfants terribles of synthetic biology, opened his spectacular talk at Sci Foo with a slide of Freeman's article. I can't express the degree to which I admire Freeman. Among other things, he was the one who turned me on to an amazing 11-sided geometric figure (see Jaron's World, April 2007). In this case, though, we see things differently.

Freeman equates the beginnings of life on Earth with the Eden of Linux. Back when life first took hold, genes flowed around freely; genetic sequences skipped around from organism to organism in much the way they may soon on the Internet. In his article, Freeman derides the first organism that hoarded its genes as "evil," like the nemesis of the open-software movement, Bill Gates. Once organisms became encapsulated, they isolated themselves into distinct species, trading genes only with others of their kind. Freeman suggests that the coming era of synthetic biology will be a return to Eden. Species boundaries will be defunct, and genes will fly about, resulting in an orgy of creativity.

But the alternative to open development is not necessarily evil. My guess is that a poorly encapsulated, communal gloop of organisms lost out to closely guarded species for the same reason that the Linux community didn't come up with the iPhone: Encapsulation serves a purpose.

Let's say you have something complicated like a biological cell, or even something much less complicated, like a computer design or a scientific model. You put it through tests, and the results of the tests influence how the design will be changed. That can happen either in natural evolution or in a lab.

The universe won't last long enough to test every possible combination of elements in a complicated construction like a cell. Therefore, the only option is to tie down as much as possible from test to test and proceed incrementally. After a series of encapsulated tests, it might seem as though a result appears magically, as if it couldn't have been approached incrementally.

Fortunately, encapsulation in human affairs doesn't need lawyers or a tyrant; it can be achieved within a wide variety of political structures. Academic efforts are usually well encapsulated, for instance. Scientists don't publish until they are ready, but publish they must. So science as it is already practiced is open, but in a punctuated way, not a continuous way. The interval of nonopenness—the time before publication—functions like the walls of a cell. It allows a complicated stream of elements to be defined well enough to be explored, tested, and then improved.

The open-source software community is simply too turbulent to focus its tests and maintain its criteria over an extended duration, and that is a prerequisite to evolving highly original things. There is only one iPhone, but there are hundreds of Linux releases. A closed-software team is a human construction that can tie down enough variables so that software becomes just a little more like a hardware chip—and note that chips, the most encapsulated objects made by humans, get better and better following an exponential pattern of improvement known as Moore's law.

The politically incorrect critique of Freeman's point of view is that the restrictions created by species boundaries have similarly made billions of years of natural biology more like hardware than like software. To put it another way: There won't be an orgy of creativity in an overly open version of synthetic biology because there have to be species for sex to make sense.

I seem to hold a minority opinion. I've taken a lot of heat for it! I can't hire Martha Stewart as a life coach, so the one thing I hope synthetic biology won't import from the open-software world is the cultlike mania that seems to grip so many open-source enthusiasts.

Published by http://discovermagazine.com/2007/dec/long-live-closed-source-software