People who are serious about wetware should write their own software

Kunal Mehta

26 January 2014

> *"People who are really serious about software should make their own hardware."*
>
> —Alan Kay, in a talk in 1982

The F-35 is America's next general-purpose jet fighter, a $1.5 trillion program that is hundreds of billions of dollars over-budget and many years behind schedule. The biggest contribution to this delay is the development of the most sophisticated software ever written for an airplane. Lockheed recently built a $150 million laboratory and hired two hundred additional engineers to finish this software, which will control flight computers, spy equipment, and an array of sensors that will enable the pilot to identify and fire at targets regardless of the direction the plane is pointing in. It has ten million lines of code (over twice as many as America's last fighter, the F-22, which is in some ways even more capable than the F-35), and it is at the heart of the airplane.

Practical electric cars are a widely-held goal that has eluded us for decades. The best ones today are built by Tesla Motors. One of the ways in which Tesla solved the problem of getting both useful range and impressive performance is by writing software to precisely control the discharging of the battery, so that the available power is used optimally. Elon Musk has cited Tesla's location in Silicon Valley – and the respect for software that comes with it – as one of the reasons why Tesla is the company leading us to this future.

Everyone knows software is important, and lots of companies have done well creating all kinds of pure software products, from the insipid to the revolutionary. But we have also gone beyond that. When Steve Jobs introduced the iPhone, he said of the user interface, almost as an afterthought, "...and, of course, it's an interplay of hardware and software". There is a class of software which exists not as an end in itself but at the heart of some other thing: a car, a plane, a phone. And most of the time, if it's viewed as more than a dumb functionalizing component, it can elevate the thing to a qualitatively new level of greatness. It can mean the difference between a Leaf and a Tesla, a phone and a smartphone, the F-16 and the F-35.

Which leads to Alan Kay, and biology. Biology was the last of the sciences to become quantitative and applied, and it seems biological engineering will be the last of the engineering disciplines to be really transformed by software. But that transformation will happen, and the potential for it exists now. It has already occurred in genomics, for example: without sophisticated data analysis software, it simply would not be possible to sequence a human genome, even though we can sequence smaller fragments of DNA and sequencing a human genome is no different in terms of chemistry.

Part of this transformation will be to use software to automate and streamline things that are currently done by hand. Software will be used to aid in designing products: to help decide what genes or versions of a gene to use in a pathway, predict the performance of engineered organisms, or make it possible to design genetic constructs on the genome scale. In this stage software will help to improve the process aspects of biological engineering.

I think some of the most exciting projects in this area are efforts to build computational models of entire cells (the Markus Covert group at Stanford and, to a lesser extent, the Bernhard Palsson group at UCSD) and do detailed atom-level simulations of protein folding (the "Anton" computer developed by D. E. Shaw Research).

But software can also be a part of an overall bioengineered product, in the same way that software is a part of a Model S or iPhone. This exists to some extent: feedback programs are used to control large-scale cell cultures in the biotechnology industry, for example. Beyond this, I must admit to having a only a dim idea of what software integrated with biology will look like. But I believe delivering the promise of biotechnology to society will increasingly require us to be as willing to write software as we are to build wetware.