The New York Times | https://www.nytimes.com/2016/05/22/opinion/sunday/to-write-software-read-novels.html

OPINION

To Write Better Code, Read Virginia Woolf

By J. Bradford Hipps

May 21, 2016

Mountain View, Calif. — THE humanities are kaput. Sorry, liberal arts cap-and-gowners. You blew it. In a software-run world, what's wanted are more engineers.

At least, so goes the argument in a rising number of states, which have embraced a funding model for higher education that uses tuition "bonuses" to favor hard-skilled degrees like computer science over the humanities. The trend is backed by countless think pieces. "Macbeth does not make my priority list," wrote Vinod Khosla, a co-founder of Sun Microsystems and the author of a widely shared blog post titled "Is Majoring in Liberal Arts a Mistake for Students?" (Subtitle: "Critical Thinking and the Scientific Process First — Humanities Later").

The technologist's argument begins with a suspicion that the liberal arts are of dubious academic rigor, suited mostly to dreamers. From there it proceeds to a reminder: Software powers the world, ergo, the only rational education is one built on STEM. Finally, lest he be accused of making a pyre of the canon, the technologist grants that yes, after students have finished their engineering degrees and found jobs, they should pick up a book — history, poetry, whatever.

As a liberal-arts major who went on to a career in software, I can only scratch my head.

Fresh out of college in 1993, I signed on with a large technology consultancy. The firm's idea was that by hiring a certain lunatic fringe of humanities majors, it might cut down on engineering groupthink. After a six-week programming boot camp, we were pitched headfirst into the deep end of software development.

My first project could hardly have been worse. We (mostly engineers, with a spritzing of humanities majors) were attached to an enormous cellular carrier. Our assignment was to rewrite its rating and billing system — a thing that rivaled maritime law in its complexity.

I was assigned to a team charged with one of the hairier programs in the system, which concerned the movement of individual mobile subscribers from one "parent" account plan to another. Each one of these moves caused an avalanche of plan activations and terminations, carry-overs or forfeitures of accumulated talk minutes, and umpteen other causal conditionals that would affect the subscriber's bill.

This program, thousands of lines of code long and growing by the hour, was passed around our team like an exquisite corpse. The subscribers and their parent accounts were rendered on our screens as a series of S's and A's. After we stared at these figures for weeks, they began to infect our dreams. (One I still remember. I was a baby in a vast crib. Just overhead, turning slowly and radiating malice, was an enormous iron mobile whose arms strained under the weight of certain capital letters.)

Our first big break came from a music major. A pianist, I think, who joined our team several months into the project. Within a matter of weeks, she had hit upon a method to make the S's hold on to the correct attributes even when their parent A was changed.

We had been paralyzed. The minute we tweaked one bit of logic, we realized we'd fouled up another. But our music major moved freely. Instead of freezing up over the logical permutations behind each A and S, she found that these symbols put her in the mind of musical notes. As notes, they could be made to work in concert. They could be orchestrated.

On a subsequent project, our problem was pointers. In programming language, a pointer is an object that refers to some master value stored elsewhere. This might sound straightforward, but pointers are like ghosts in the system. A single misdirected one can crash a program. Our pointer wizard was a philosophy major who had no trouble at all with the idea of a named "thing" being a transient stand-in for some other unseen Thing. For a Plato man, this was mother's milk.

I've worked in software for years and, time and again, I've seen someone apply the arts to solve a problem of systems. The reason for this is simple. As a practice, software development is far more creative than algorithmic.

The developer stands before her source code editor in the same way the author confronts the blank page. There's an idea for what is to be created, and the (daunting) knowledge that there are a billion possible ways to go about it. To proceed, each relies on one part training to three parts creative intuition. They may also share a healthy impatience for the ways things "have always been done" and a generative desire to break conventions. When the module is finished or the pages complete, their quality is judged against many of the same standards: elegance, concision, cohesion; the discovery of symmetries where none were seen to exist. Yes, even beauty.

To be sure, each craft also requires a command of the language and its rules of syntax. But these are only starting points. To say that more good developers will be produced by swapping the arts for engineering is like saying that to produce great writers, we should double down on sentence diagraming.

Here the technologists may cry foul, say I'm misrepresenting the argument, that they're not calling to avoid the humanities altogether, but only to replace them in undergraduate study. "Let college be for science and engineering, with the humanities later." In tech speak, this is an argument for the humanities as plug-in.

But if anything can be treated as a plug-in, it's learning how to code. It took me 18 months to become proficient as a developer. This isn't to pretend software development is easy — those were long months, and I never touched the heights of my truly gifted peers. But in my experience, programming lends itself to concentrated self-study in a way that, say, "To the Lighthouse" or "Notes Toward a Supreme Fiction" do not. To learn how to write code, you need a few good books. To enter the mind of an artist, you need a human guide.

For folks like Mr. Khosla, such an approach is dangerous: "If subjects like history and literature are focused on too early, it is easy for someone not to learn to think for themselves and not to question assumptions, conclusions, and expert philosophies." (Where some of these kill-the-humanities pieces are concerned, the strongest case for the liberal arts is made just in trying to read them.)

How much better is the view of another Silicon Valley figure, who argued that "technology alone is not enough — it's technology married with liberal arts, married with the humanities, that yields us the result that makes our heart sing."

His name? Steve Jobs.

J. Bradford Hipps, a former developer who works in software, is the author of the novel "The Adventurist."

Follow The New York Times Opinion section on Facebook and Twitter (@NYTopinion), and sign up for the Opinion Today newsletter.

A version of this article appears in print on , Section SR, Page 7 of the New York edition with the headline: To Write Software, Read Novels