Questions and answers on "JavaScript Affogato: Programming a Culture of Improvised Expertise"

Brian Shea and Brian Lennon

Published by Johns Hopkins University Press Blog, 28 March 2018: https://www.press.jhu.edu/news/blog/program-resumed.

A Q&A about the essay "JavaScript Affogato: Programming a Culture of Improvised Expertise," published in *Configurations* 26.1 (2018): 47–72, DOI: 10.1353/con.2018.0002.

Brian Shea: How did this article come about for you?

Brian Lennon: This article is the first publication drawn from my current research on programming languages and cultures of software development, work in which I take a quite specific type of humanities-based approach (one that isn't necessarily compatible with other humanities-based approaches of our moment). In some ways the project represents a return to research in the cultural history of computing, and specifically computer programming, that I'd planned to pursue as long as twenty years ago. Those plans were deflected by the so-called dotcom crash of 2000–2002 and the concurrent events of September 2001, which fairly quickly made other topics and issues more urgent to me. It seemed at the time that for all of its problems, the Silicon Valley-centered tech industry culture of the 1990s boom had gotten more or less what it deserved when the crash came, and that history's verdict was enough. I don't think I expected at the time that this wouldn't (yet) be the end of that particular story: that the national security industry created after September 2001, channeling vast resources into new modes and scales of automated data collection and analysis, would offer both surviving companies (e.g., Amazon, Google) and new companies (e.g., Facebook, Twitter) a new vector for growth — while the U.S. and world economic crisis that began almost exactly seven years later offered them outright hegemony: that is, cemented their image as what Gideon Lewis-Kraus recently called "the last redoubt of confidence and productivity in an otherwise uneven recovery." In that sense this project represents interrupted or otherwise unfinished business, though I'd thought I'd finished with it in 2001.

Brian Shea: How important is it to examine the development of something like JavaScript, which most people have probably heard of, but have no idea

how it has affected their lives over the past 20-plus years?

Brian Lennon: Today it is common to hear that software runs the world, that life as we know it is impossible without software, or that software is "eating the world" (not necessarily as a good thing, but certainly as an inevitability) and so on. One even finds scholars and cultural critics in my own area (the humanities, broadly speaking) echoing such pronouncements — sometimes explicitly as a justification for directing attention and resources toward so-called software studies as a non-technical, that is to say primarily historical and cultural research area. While such pronouncements aren't untrue, they are often what I'd call marginally dishonorable, in two ways. The first problem with such claims is that much of life as we know it has been running on software for nearly seventy years already. There is, in other words, nothing especially new here. One does not hear similarly aggrandizing claims about nuclear energy, mass-produced plastics, color television, or other indispensable and ubiquitous technologies whose emergence also dates to the 1940s. The second problem with a pronouncement like "software runs the world" is that it directly serves both the economic advantage and the generalized economic, political, and cultural authority of a very specific kind of technical expert, the computer programmer, whose economic role (with those of other experts who directly support or exploit the programmer's work) has been elevated beyond reasonable measure. Though their work conditions aren't perfect and they gripe like anyone else, professional programmers were uniquely unscathed during an era of austerity and generalized economic pain and suffering, and many are insensitive to the context of their good fortune. I can't find it in me to join the chorus here, given how nakedly such talk reflects the extensive economic violence of the interval since 2008, a period in which "learning to code" has been imagined — sometimes confusedly, but often quite cynically — as something like a universal pathway to re-employment.

One cannot deny the importance of software. But for me there's a meaningful difference between the circular reasoning of such pronouncements — "software is important; important things deserve attention; therefore software deserves attention" — and the historical, economic, and political questions of *how* and *why* software came to be so important, along with the normative question of whether software's importance is, on balance, something good or something bad. As a programming language designed for accessibility to both novice programmers and non-programmers that outgrew its design unexpectedly and violently, JavaScript illustrates the typical and unsurprising refinement of most forms of technical expertise over time, in a process that leads to full but narrow specialization. That is to say that while the tempo of that dynamic in this case makes for good illustration, the dynamic itself is entirely typical. While I don't expect this issue to be resolved in my lifetime, I think it's silly to expect computer programming to prove as generalizable a technical skill as those associated with reading and writing (that is, conventional mass literacy) or in earlier epochs those practiced in, say, subsistence hunting and agriculture, or paleolithic architecture. The better analogy is to a technology like the automobile. Given the many other distinct and equally valuable forms of expertise on which life as we know it relies, it's as ridiculous to imagine *everyone* knowing how to build a software application as it is to imagine *everyone* knowing how to build an automobile. That's why I regard the nostrum "learning to code" as mostly cynical: if I told you that in order to be employable in almost any healthy segment of the world economy today, you'd need to know not only how to repair your own car, but how to build one from scratch, you'd rightly suspect I was up to something.

Brian Shea: How surprising is it to you that studies of computer programming culture is seemingly underrepresented in the humanities?

Brian Lennon: It isn't surprising to me at all, but here too my reasoning is inassimilable to what one tends to hear elsewhere. I'm one of a small group of contemporary humanities-based researchers whose specific attitudes on this issue — neither reactionary nor opportunistically, performatively progressive, but informedly critical — are mostly illegible in public discourse today, though that is absolutely not for lack of all kinds and depths of conventional social privilege. It's not at all an issue of attention; it's an issue of understanding, and to some extent of motive, disposition, and of course economic investment. I came to humanities research deliberately, as someone with both scientific and technical aptitudes but no *problem* with the humanities or with the entire existing structure of disciplines, which I regard as having developed historically, for historical reasons, and not willable away in the name of facile interdisciplinarity. Unlike many of my professional colleagues, I'm uncowed by the authority of technical expertise, and I don't find it magical; also, I'm not asking the humanities, either explicitly or implicitly, to be something other than what they already are. If I'd wanted to go into the social sciences instead, I'd have done that. The same for the technical sciences. Like those of anyone otherwise free to align himself with new sources of economic power and authority, my reasons for deliberately choosing the humanities may deserve to be called eccentric. But they are what they are; conscience is too strong a word, but hesitation before what is easy and available by birthright might not be.

It's true that people who know a lot about computing generally don't take up humanities research and teaching as a career. The exceptions sometimes seem conflicted about their choice, though perhaps only for purely personal reasons; either way, too often in such cases, what I see is someone converting an unusual depth of technical knowledge to a dishonorably ideological advantage in a specific institutional context. I saw quite a bit of that during the 2008 economic crisis, in the emergence of an ideologically aggressive and distinctly messianic "digital humanities" movement presenting itself as an opportunity for the recipients of useless undergraduate or graduate degrees in subjects like English, modern languages, history, and philosophy to redeem their poor educational choices *and* magically vanquish the problem of un- and under-employment by learning to code.

The only thing that surprised me at first, in such developments, was that the attention of humanities researchers — that is, experts in the study of both text and language — drifted rapidly toward computational statistical analysis and the applied mathematical substrate of computing, in the embrace of so-called cultural analytics trained on masses of digitized text. This was a disastrous choice, I'd say, given that for perfectly good and not at all lamentable reasons, both aptitude and interest in even basic applied mathematics is less common in the humanities, and more difficult to sustain, than an affinity for computing more broadly conceived. When you consider the role that applications of advanced research in topology and graph theory play in both the infrastructure and the *imaginaire* of big data, today, humanities researchers' embrace of the association suggests nothing more or less than panic and grasping at straws — not waving, but drowning. If this is surprising, it's because other options were available. For example, given the historical contiguity of linguistics (a humanistic and social-scientific discipline in equal measure) and the technical domain of programming language design and theory, one might have expected more of us to focus on that contact zone well within our reach. While programming language theory is hardly removed from mathematics, the role of mathematics is mostly historical (if we accept, for example, that the roots of PLT are in the lambda calculus), and I'd say that a deeply and thoroughly trained humanities researcher, even one with little organic affinity for computing, already possesses the larger part of the expertise needed to take a programming language as a research object in itself — rather than as a research means or tool, applied casually to inexpert, possibly poorly understood, more or less obliquely arithmetical ends. There is, in other words, no need and no question of retraining for ostensibly more pertinent work.

Brian Shea: Why did it make sense to publish in *Configurations*?

Brian Lennon: *Configurations* is a journal publishing research of high quality in a productively squishy area at the intersection of literary scholarship, science and technology studies, and the arts. It's distinguished itself in that location for twenty-five years. While it's hospitable to the kind of research I'm doing now, it has kept both the weaker claims of so-called software studies and the puerile excitability of the "digital humanities" movement at arm's length. I keep reading it for that reason, among others. It happens that my very first research article, on a not dissimilar topic, was also published in *Configurations,* nearly two decades ago; and while I had no problem imagining my more recent work also fitting in, when I searched the archive of back issues before submitting "JavaScript Affogato" to the journal's current editors, I found that I myself was the only contributor to have previously mentioned JavaScript in its pages!