Johnson, Nathan R. and Damien Smith Pfister. 2013. "'Ecologizing' Berry's Computational Ecology."
Social Epistemology Review and Reply Collective 2 (4) 7-9.
http://wp.me/p1Bfg0-IA

"Ecologizing" Berry's Computational Ecology

Nathan R. Johnson, Purdue University and Damien Smith Pfister, University of Nebraska-Lincoln

David M. Berry's "The Social Epistemologies of Software" can be profitably read in a number of ways: as a rich explanation of how the hyper-reflexivity of networked software underlines the intense sociality of computational knowledge formation; as a careful account of social software at the cusp of pervasive computing and the internet of things; as an effort to publicize the ironically similar phenomena of "web bugs" and "lifestreaming;" as a first step in theorizing what he calls "compactants," the "computational actants" that monitor behavior, geolocation, affects, and more; and as a warning about the under-appreciated normative dimensions of screenic representations and the computational care of the self.

It is hard to argue with the premise that software's influence is increasingly ubiquitous. Similarly, the notion that scholars and citizens ought to turn their critical faculties toward code is similarly unimpeachable. If, indeed, "code and software [have] become the conditions of possibility for human living, crucially becoming computational ecologies," then we must take them seriously (Berry 2012, 379). Berry's hope is that more critical attention to code and software will expose how contemporary social epistemologies of software are constituted and thus enable "intervention, contestation, and the *unbuilding* of code/software" (Berry 2012, 393). Though the "techniques needed [to understand code/software] are still in their infancy," they will necessarily require a multidisciplinary effort (Berry 2012, 392).

Precisely because the analytical approach to software and code is so new, the assumptions undergirding any interpretive effort deserve special attention. Thus, in this critical reply, we want to amplify the ecological in Berry's "computational ecologies" by drawing on Jenny Edbauer's work on "rhetorical ecologies." An "ecological approach to computation," rather than a study of "computational ecologies," might provide a useful way of generating insight into how the social epistemologies of software are formed. Berry's work seems amenable to this extension, given his professed goal to "understand software ecologies as a broad concept related to the environmental habitus of both human and non-human actors" (2012, 380-1). Our inflection, though, differs slightly: an ecological approach asks not what "knowledge [is] represented in code and software to better understand software ecologies," but what micropractices were constitutive of that knowledge framework (Berry 2012, 391).

Edbauer's notion of rhetorical ecologies is developed in contrast to more static theories of "rhetorical situations." When trying to describe a rhetorical situation — a speaker before an audience, a blogger preparing a post — critics have attempted to break down the moment into component parts. "We find in the early models of rhetorical situation," Edbauer notes, "a notion of rhetoric as *taking place,* as if the rhetorical situation is one in which we can visit through a mapping of various elements: the relevant persons, objects, exigence, and utterances" (2005, 12). This approach to rhetorical situations as merely "elemental conglomerations" (2005, 7) elides the fact that "we are never outside the

networked interconnection of forces, energies, rhetorics, moods, and experiences. In other words, our practical consciousness is never outside the prior and ongoing structures of feeling that shape the social field" (2005, 10). Of course, analytical precision often requires a parsing of elements that unavoidably does an injustice to the richness of experience; Edbauer's critique cautions us to beware when this analytical separation obviates a sense of rhetoric's *dunamis,* or potentiality. Thus, "an ecological augmentation adopts a view toward the processes and events that extend beyond the limited boundaries of elements" (2005, 20). Edbauer's notion shares affinities with several other modern schools of thought within the social sciences, especially those following in the wake of Bourdieu, Giddens, or Latour.

There is a risk in analyzing software, just as in analyzing traditional modes of rhetoric, that complex processes will be reduced to conglomerations of elements. Web bugs track users. Users discipline themselves by quantifying their activities. While Berry is attuned to the interactional processes functioning between code, objects, and human subjects, his analysis of computational ecology could be pushed even further towards an ecological orientation. Here, let us identify two places where this is possible.

The focus on surveillance, replete with metaphors of tracking and bugging, occludes attention to broader and more foundational software epistemologies. Bugs and beacons, though important subsets of software, comprise a relatively small portion of software practices. Indeed, bugs and beacons could be understood as an epiphenomenon of more infrastructural software, software so central and pervasive in modern computation that it functions outside of critical perception, and hence is more difficult to analyze. Here we imagine something like the classification software organizing vast databases of information. Without this more foundational software, is there any code to bug? For instance, by itself, a bug or beacon says little about epistemologies mingling with a gendered catalog of Amazon items or fitness practices that enable lifestreaming. Just as biologists could not understand a parasite without analyzing the host, so cannot critics of software analyze "bugs" absent an analysis of a broader view of software practices. We would note, too, that the framing of these pieces of code as "bugs" risks overdetermining analysis in one particular direction—much in the same way that the overbroad invocation of "parasite" masks relationships based in commensalism. Part of a critical theory of software ought to involve the invention of a novel vocabulary to describe software epistemologies capable of moving beyond our current nomenclature (though such a move risks depoliticizing code, as when Berry notes that the industry-preferred term for web bugs is "clear GIFs.")

Taking an ecological view draws attention to the instability of another set of terms that Berry employs: the binary concept of user/programmer. At first blush, these terms imply that some individuals have more control of software than others, that perhaps they understand enough to manipulate their own bugs, beacons, and lifestreams. This contrast, though, surely isn't as simple as a Foucauldian disciplinary surveillance in which the super programmer dominates the user. Upon deeper inspection, the distance between the user and the programmer seems less absolute. The different actions of a user shape the functioning of the code because of the algorithmic processing built into software.

Facebook algorithms detect who you interact with the most and push their status updates to the top of the feed. Google News finds out what kinds of stories you like and filters daily happenings to match your interests. On the flip side, programmers cannot be cordoned off as a separate entity—for they are users too! Neither user nor programmer are ontologically given; they are constantly in a process of becoming.

An ecological approach to computation might, then, explore the kinds of affects that are attempting to be mobilized by the invocation of "programmer" and "user" by participants in software design and use. Perhaps instead of suggesting an elite group of programmers are collecting and analyzing data, we might start a critical theory of code by asking: what does the creation of bugs say about the various individuals touched by the software? For a programmer, we might consider what the act of creating bug software means. Could it be that bugs are traces of social positioning: that bugs are an effect of individuals marking themselves as programmers in contrast to users? If that is the case, what epistemological notions blur that distinction between programmers/users, and what does software say about each? We might also ask how does computation reinforce existing social practices of computational individuals? Or perhaps we might ask, does using software that is known to track personal behavior a method of *playing* with the infrastructures of a modern software society? Users frequently know their behavior modifies tracking algorithms even as it alters their own behavior. But they frequently behave as though their personal data is hidden from others, too minor to be noticed in the seas of "big data" or create multiple data selves difficult to instrumentally pin down, enumerate, and control.

Perhaps then, the practice of code criticism should begin by studying it in its local micropractices, casting off preconceived notions of users/programmers, bugs/beacons/lifestreams, and code/content. A critical theory of code might start by asking: what ecologies of computation are discovered through the practices and *intentionalities* of code? To further explain, what are the ways in which individuals articulate and legitimate their computation while simultaneously participating in a regime of ongoing algorithmic practices beyond the explanation of any one person?

Contact details: **nrjohnson@purdue.edu, dpfister2@unl.edu**

References

Berry, David M. 2012. "The Social Epistemologies of Software." *Social Epistemology* 26 (3-4): 379-398

Edbauer, Jenny. 2005. "Unframing Models of Public Distribution: From Rhetorical Situation to Rhetorical Ecologies." *Rhetoric Society Quarterly* 35 (4): 5-24.