Journal of e-Media Studies
Volume 3, Issue 1, 2013
Dartmouth College

Matthew Fuller in Conversation with Mark Marino

Mark Marino: Even though the context of our chat is this special issue, this is a conversation we've been wanting to have for a while, and it's nice to have an opportunity to have it in this publication.

All right, Matthew, the first question I have for you is, over the past ten years since you called for software criticism, how do you feel overall that software studies itself has evolved?

Matthew Fuller: I think I'd want to avoid any kind of originary claim. I think that software studies is just something that emerged from many different people and many different practices carrying roughly around the same kind of time.

Marino: Now, let me interrupt there and say, but at some point, some aspect of software studies wasn't happening the way you expected it to, though. I'm thinking back to your article where you say, essentially, "We kind of need to put this critical attention here." Again, was there something lacking, was there something missing?

Fuller: Yes, for sure. I think exactly it's not to say that things weren't missing, but what I would draw attention to was a broader movement. I think a lot of the impetus for software studies has come out of software art, came out of interface design, came out of hacking practices, came out of free software, came out of computer scientists talking about culture and developing a relationship to culture and so on.

Also, scholars in computing history have over a long period of time developed varied kinds of engaged accounts of computing and its wider ramifications. Then of course, the kinds of developments that have come through things like STS and so on, and all of these kinds of currents converged in a sense on the problematic of software. We can also say media theory, specifically in some of the iterations occurring in the Germanophone areas, have also crystallized certain approaches to technology, which has also driven a lot of the more interesting and attentive work in software studies.

If we look at, for instance, the way in which geography has, as a discipline, begun to think through software, this has also made major contributions to the discussion—through the work of people like Steven Graham, Martin Dodge, and Rob Kitchin. I think software studies has become an area that is really vibrant and interesting, because it's a space in which people are coming together with lots of different kinds of expertise, and they're willing, and need to, take the time to pay attention to work coming from different disciplinary backgrounds with different kinds of research and practice ethos.

I guess if the question is how is it changed, it's really that also the domain has become increasingly rigorous. When you think of Wendy Chun's recent book [*Programmed Visions: Software and Memory*, 2011], both the kind of depth of historical research that goes on there and the kind of attentive speculation that she's working through really sets a strong precedent.

Marino: Yes, but it's very unusual to have this electrical engineer trained in continental philosophy. Is she the sign of a new type of scholar, or do you feel that this is just a natural progression as cultural studies scholars become more computationally literate and vice versa?

Fuller: Wendy is someone who really sets the pace in many ways, but this is something that can also be seen in terms of a newer generation with particular kinds of combinations of knowledge from growing up in and driving the present era. I think there are a number of things happening. One is that people also begin to collaborate more, so you get collaborations between people with different kinds of knowledge, more technical, more cultural, political knowledge, and experience. You get also people coming from cross-pollinated backgrounds from computing into more cultural domains. And as computing becomes increasingly something that relies on the social, relies on culture in order to drive it—games, social media, logistics, control and coordination mechanisms, and so on—the problems that computing and culture share are some of the key problems at the present moment.

If you take some of the most mainstream examples of the development of computing, such as Facebook or Google, the problems that they're facing are in a way many of the problems that cultural studies and so on have been looking at for many years. The ways in which they tackle these problems are fundamentally different in many respects, indeed antithetical in some, but there are also ways in which we have to understand culture and the social ramified through its increasing interpolation via computing, which means that at the same time, computing itself changes.

This is a very interesting moment, I think; a lot of computing is now done in and through the social, and a lot of culture is now carried out or executed in computational environments as they are also, in turn, changed by their involvement in space, cities, systems of semiosis, and so on. This kind of conjunction is the real richness of the present moment.

Marino: Before the interview, you and I were talking about some of these collaborations across the disciplines, and I feel like I have encountered a dynamic where amongst those in—I'm not even going to say computer science—amongst those who build, there is one set of critical practices and approaches. And amongst those who interpret, there's often a level of suspicion which they apply to all of these technologies.

To some extent, when I have conversations with people who primarily, for example, teach computer science, I find divergences in the nature of our critical approaches. Again, not that there isn't a kind of suspicion that happens when someone considers a piece of

code that someone has written, evaluating it from the standpoint of trying to get them to become better programmers. It's very, very different, however, from the kinds of questions that I ask of most of my objects of inquiry.

Again here, in the backdrop of all of this conversation we're having right now, I'm thinking about your new forthcoming book project, *Evil Media* [MIT Press 2012], again, which is challenging the fundamental "do no harm," "we're the good guys" mantras of contemporary computer software companies.

These engineers and programmers may even, by their lights, be working towards making technology objectively "better." Such rhetoric makes sense in terms of marketing but, even more importantly, makes sense when you are in the business of engineering things that are more efficient, take up less memory, et cetera. To some extent, that critical practice butts up against those who are searching for the underlying logic of capitalism; the operations of the State; the operations of the technologies that may be determining various aspects of human interaction from the standpoint, again, of distrust, of skepticism.

That was a broad and wide question, so take any part of it you'd like.

Fuller: Yes, it's a wide landscape to construct a walk through, I guess. I think the approach I would tend to have to it would be that I don't think there's any need to negate either of those kind of positions. It's entirely understandable that people who are working

on engineering projects subject them to the kind of criteria within which success is evaluated in engineering as a discipline.

You have the scope of a particular problem to solve, which may be of many different orders, often around the kind of questions of efficiency, good functionality, and so on. I think it's entirely understandable that people would want to inhabit that space and work it through to the best of their abilities and to the best of the abilities of the organizations that they're working with. To propose something else, or that a wider context or set of criteria is also taken into account, is often to evoke suspicion. But at the same time, there are many other conditions in which things aren't that stable.

Certainly, if you look at some of the early histories of computer science, when it was emerging as a discipline in the '40s, '50s, and '60s, things weren't that stable. Or at what it involved if you look at the early writings of people like Dijkstra or Knuth—all of these people that founded the discipline, or the founding of object orientation for instance, coming in part out of the trade union movement. I think we are going through a particular point at the moment where a renewed understanding of the underlying problematics of computing, and its relationship to the problems it tries to solve and the way it formulates these problems, can be made. The kind of resources it brings today on those problems can be evaluated in a way that's fruitful both from engineering perspectives and from more broadly cultural or software studies perspectives.

That's what I'm trying to say, rather than kind of ramify the differentiation between these fields, is trying to inhabit or trying to formulate the way in which the central problematics in computing are changing in the present moment in relationships to the problems they solve or that they're trying to address, at least. And in such inhabitation, see what kinds of conjunction of interest can be produced. I think that's a key thing that software studies is doing at the moment, and you can see this in many different projects producing code, hardware, software, and also in the different theoretical lines being explored.

Marino: What are some examples of areas or projects using the conjunction of interest that you were just describing?

Fuller: I mean I think this is something we in the Center for Cultural Studies, at Goldsmiths [College, University of London] take as a kind of starting point for thinking through the development of software, and through working in the interactive media degree and the creating social media degree that we run jointly with the computing department. Alternately, for instance this weekend, when I step outside of Goldsmiths, I'll be taking part in an event run by the company that's now called Cosm—previously Pachube—where they're looking at the "internet of things." This is a technology-driven company, which runs on the basis of providing an infrastructure and a markup language for communication in the internet of things.

Their focus in this event is to try and think through what would be an adequate bill of rights for data, for sensors, for data-subjects, and so on in the present era. At the moment,

when computation is expanding into the social, into the urban, into the interrelation between ecologies and entities, and objects, processes, people, and so on, and at the same time, technology is increasingly interpolated by law, what formulations are necessary and what kind of languages can be deployed to think through and enhance the technology? These kinds of points, where technologies are developing standards, technologists are developing a means or are developing means of talking about the outside world and about the world they inhabit, these are kind of crucial and very interesting moments.

Marino: Let me follow up with the request. Could you give us some advice or act as a kind of guide, give some sense how someone who's approaching technology studies from the point of view of software studies can avoid—again, well, to me, it's kind of a trap that can happen to some degree of just becoming a new formalism. A kind where media archeology is put implicitly into the service of the technology industries, building exhibits for their museums which tend to not necessarily celebrate their victories, but certainly ensure their preservation. Of course, industry welcomes this approach, and then I think we welcome it, too, in terms of our nostalgia. Still, even though such an attitude may not short-circuit our critical practices, it gives them short shrift.

Fuller: That's a very interesting problem.

Marino: I feel like again, somewhere, it's in the framing of the question, but I'm not sure because too often it seems like the value that comes out of software studies, platform studies, code studies, is that artifact or that aspect of the artifact that we've detected by

putting it under our electron microscope. It's valid because it contributes to real scientific knowledge more so than it provides cultural insight necessarily.

Fuller: I think there's many ways in which to avoid that trap. I think it's something of a challenge to find a means of feeling entirely persuaded with what's currently settled out as a society. It's difficult to find a way of ratifying it in so many respects, so I think that kind of questioning is somewhat inherent to being alive in the present time, paying attention to the weirdness of contemporary life in the way that some of its most highly potent aspects and institutions, from politics to economics, to statistics, media, police, and so on are going through different kinds of failure and crises.

I think the question of what one comes with to the technical object, the technical system, is always going to be to a certain extent asymmetric to an idea of confirming its nature as natural. At the same time, I think there are ways in which studies of technology and software cultures can sometimes simply confirm what is known, can simply document or act as a kind of extended press release to the standard image of software.

If you come to the question of technology or programming or computing with any sense of its history, with any sense of what's included and what's excluded from it, then I think it's most likely that some kind of disruption of that normed account is enacted. The question really, then, is to try and build encounters between different fields, between different kinds of methodologies, and also to try and build infrastructures whereby such

kinds of research can be maintained and developed. Hence, for instance, the wide interest we see now in building journals, and other kinds of platforms.

Marino: Let me change for a second to the topic of code, something that you and I share an interest in. First of all, do you feel like with the advent of 2012 "code year," and people like Mayor Bloomberg of New York saying he's going to learn how to program, and the rise of other sites where people are learning how to program on their own, and of course, the development not just of things like Scratch and Snap, but also the development of Processing and many other languages . . . are we seeing a renaissance of the dream that was started with BASIC, or is this a cultural shift towards a desire to learn how to program?

Fuller: Yes, it's pretty interesting, isn't it? I think anything that generates, in someone like Michael Bloomberg, a scintilla of doubt about the kind of magnificence of his knowledge is certainly to be encouraged. If he were to take learning code seriously, it would probably take him several years, which would no doubt be a great benefit to humanity if he were to take that on and depart from his present role.

I mean, in the UK, I guess a parallel initiative is the Raspberry Pi computer, which is a very cheap computer costing about, say, twenty-five pounds, about forty dollars, but based on an ARM chip, and is geared towards kids learning programming in Scratch and Python and so on. It runs Debian or Fedora and is a beautifully designed piece of hardware that's developed by open hardware enthusiasts supported by different

universities, Cambridge particularly, and has come out of a kind of sense that the advantages of the BASIC era of programming can be revisited through the developments of free software, through languages such as Python.

It's quite a fabulous initiative that's only just begun to be publicly available in the last couple of months. It's a really super piece of hardware that's extremely cheap that is really pitched absolutely at this kind of question of reestablishing the imaginary of computing. Not simply as a kind of a must in the contemporary world, but also as a kind of space for play, for imagination, for passional fun in the sense that Olga Goriunova argues for in the Funware and Fun with Software exhibitions.

It's really about computing as a place of intellectual passion and enthusiasm. This project is really significant in the sense that it challenges you with something rather technical and quite difficult. It doesn't try to simplify things too much. At the same time, it does try to simplify things at the level of economic cost of hardware and of access. It's extremely nicely designed on those levels. It's also very much a kind of counterblast to other models of computing education in the UK. For instance, previously, the chambers of commerce and various business organizations called for school pupils to be taught what they called "Functional ICT," which is basically learning Microsoft Office programs, standard, contemporary secretarial skills.

Marino: Great.

Fuller: It has to do with spreadsheet. It has to do with processing.

Marino: It feels like there are a lot of lessons in browsing?

Fuller: Yes, this basic kind of skill that is needed to become a functional unit within contemporary lower-level office work, was put in place by the previous government. The counterinitiative comes from computer scientists, educationalists, and so on. It's a lot more inspiring and certainly seems to get children interested, but certainly could be more systematically integrated into schools.

Marino: Okay. Now, we were just talking about children, but I just got done speaking with some electronic literature scholars in Spain in the LEETHI group, where, as always, the question came up about how important or not it was for them to learn how to program. Of course, I know many literary scholars who have discounted their own ability to learn how to program. Forget children, here grown adults who have mastered multiple natural languages are asking whether or not this is something they need to do or could do. Is there something in your experience—and again, this is an area of interest of mine, of course—that someone who has a training in cultural studies, literary studies can sort of benefit from when they start to pick up a programming language? Is there something in their linguistic skills that makes them a unique kind of programmer once they learned how to program? Has all their time in books been a detriment to their need to pick up a proper programming language?

Fuller: I think it's also the question whether code is the main point of interest to software? I don't think that's entirely the case. Whilst code is a really important place to site critical attention to software, it's simply one place. You can look at the underlying logic, abstraction layers, interfaces, data, the whole question of the different kinds of forms in which computing interacts with culture and with life as being equally privileged points of access.

Marino: All right now, as you might imagine, before you go on with that, I might push back on that slightly, given my own priorities. One thing I worry about when we don't examine the code layer is that people work with abstractions that aren't, for example, accurate to what is happening at the layer of the program's code and how it's operating.

Fuller: Yes.

Marino: I think I certainly I appreciate that there are different points of entry, but I guess I also worry that if we avoid the level of the code, that there are complexities, particularities we'll miss. It's not that there is some super secret, hidden logic that you won't have access to. It's just that we very well just might not be correct in what we're saying.

Fuller: I think the likelihood with any statements is that there is a "proportion of correction required," let's say.

Marino: Right, right, right.

Fuller: I don't think there's any one layer of contemporary reality that is entirely privileged over the others. There are greater or lesser capacities to reveal or to produce the kind of explicit or latent at different kinds of scales. I think it would be difficult to make an argument that any one is preeminent.

Marino: Right, I understand.

Fuller: In the same way that when one wouldn't want to kind of subscribe to a vulgar Marxism, one wouldn't want to propose a vulgar computationalism—something you're not suggesting. There's a requirement for a rigorous empiricism that's clear in critical code studies, attention to the material, bringing close-reading strategies to the fore, and so on, because code is obviously crucial to software. The history of language design, for instance, with the specific predilections of each language, the kinds of statements they render crisp and tractable or verbose, the interests that underlie them, and the new interests that they make possible, has been a fascinating point of entry to the question of software. The kinds of question that attention to code makes possible are also significant, at the microscale of the individual line, or at the scale of the millions of lines that much software operates at.

Language also provides a common point of research for several fields and disciplines, which is why it's interesting to look at team forms of research in which people come with

different kinds of skills, different knowledges, and of experience that'll allow them to produce a joint problematic.

Marino: Are there strategies for helping those teams collaborate, to share their paradigms, or ways that are more or less productive? I have run into, for example, kind of deep-seated suspicion going back to the cultural wars, the science wars from programmers against the kind of critical theory that may be discussed as the lingua franca of the social critics. Of course, obviously, there are many paradigms and concepts in computer science that if you have not had, obstruct your full understanding of software. I think it's maybe also [that] certain kinds of daily headaches that the people who are trying to build better systems have to deal with are very difficult for someone who has not been involved in that practice to appreciate. Are there ways to lay the groundwork for more productive collaborations that you found over at Goldsmiths, or in the courses that have been developed, or in the collaborative projects that get started that help people work through that, the kind of acculturation, the kind of mutual teaching?

Fuller: Computing knowledge is so widespread that it's not simply held by the experts, the people who primarily develop computer science and software engineers. I think that always sets us places to start in many different fields. There are multiple legacies, running from music scenes to software art to hacking scenes, that generate different kinds of knowledge and points at which computing is actively being constructed. These are some places to start. Equally, shared problems, which are not owned by specific

disciplines like how to compute the social or how to map how the social is being computed, are also points of entry.

I think increasingly, people will and are starting to use computational methods to analyze the way in which computing is combining with culture, with the social, with politics, and that these also produce new kinds of knowledge in combining these fields. One can find such examples in work looking at social networks or at movements of information around networks, or one can see it in the kind of research done in reverse engineering of search engines and so on.

Marino: Right, it sounds like asking different sorts of questions leads to more productive collaborations in some ways. Now, and I have to ask one of the questions from the prewritten list, and one of them is, What word or words would you add to an updated *Software Studies: a Lexicon*? Or alternatively, you can answer, What's the most important word or disruptive meditation in the book? Either one.

Fuller: I think one would probably want to look at a term like *model*. In the present moment, we're living through a period in history where software is intensely active both in creating and diagnosing problems, reacting to numerous kinds of opportunities, and is also the medium through which we're understanding much of this and of the contemporary period.

One might also want to look at how something such as the model couples with another key word of the present era: *austerity*. The various ways in which it is being implemented, in part via the study and prognoses gained from certain models, wherein economics are fused with basic technocratic responses to the problems generated by the finance industry that is itself so enormously computationalized. In such a situation, we have a multilayered moiré pattern in which software is active at multiple scales, and in none of which do we have any necessary grasp on a fundamental reality without the mediation of further software. Some of the work of Donald MacKenzie points in such directions, as do some of the stratagems that Andrew Goffey and I propose in *Evil Media*. At certain times indeed, one might also want to analyze the situation via other means than software . . .

Marino: Now, Matthew, I mean, to me, that shows a complete, well, a new stage in maturation in the field because when I think back to *Software Studies: A Lexicon*, it's still asking the question, What happens when we cultural critics take on these terms straight out of technical culture? Here, you're talking about a term not from technical culture but from political culture. This term shows not conflation, but the imbrication of both or the complete inseparability of both. The questions we're asking about culture are the questions we're asking about technology. Or there are questions in culture that are infused with questions of software, right? It seems like a very different state of development than when that book first came out. Again, not to minimize what that book is trying to . . .

Fuller: The contributors to the book were all either practicing programmers or had a background in computing in some form, so it wasn't a kind of question of culture theorists picking out computing terminology from a pure space in culture and taking them back to the safe fold of theory, it's really about creating a common ground.

Marino: Right.

Fuller: Necessarily, things will move on from there. I think there's a kind of recognition in many writers, in people working in various different modes, that the questions of computation are fundamental to some of the key problems that we're facing in the world today, which is alas a very corny way of expressing it.

Thinking about the way in which knowledge of all kinds has become computational, whether it's in government, finance, all the sciences or humanities, or in straightforward experiential and social knowledge, they nearly always have an interaction with computing and the kinds of procedure it implies. All of these are reconfigured and being reshaped by computing at the same time as they, asymmetrically, shape computing. I think this is a point in time where the wide questions of politics and governance that the social imaginary as a whole takes on are being fundamentally re-inflected in the forms that computing takes—and these can go to a very low level: such as the questions of the algorithm; or the data structure; that of the abstract machine; the questions of how you formulate an entity and its relations to other entities, and so on, that characterize database design; what forms a judgment of what counts as a significant event; what counts as

something that one pays attention to, or one doesn't pay attention to, and so on. All of these are being articulated in computational grammars. Yes, I think that's a wider question to a certain extent, addressed in the new figurations of software studies, but also in the kinds of questions that the more interesting ends of the digital humanities are opening up.

About the Authors

Matthew Fuller's books include *Media Ecologies: Materialist Energies in Art and Technoculture* (MIT Press), *Behind the Blip: Essays on the Culture of Software*, and *Elephant & Castle* (both Autonomedia). With Usman Haque, he is coauthor of "Urban Versioning System v1.0" (ALNY), and with Andrew Goffey, coauthor of *Evil Media*. (MIT Press). Fuller is editor of *Software Studies: A Lexicon* (MIT Press) and coeditor of the journal *Computational Culture*, and he is involved in many projects in art, media, and software. He works at the Center for Cultural Studies, Goldsmiths College, University of London (http://www.spc.org/fuller).

Mark Marino is a writer and scholar of digital literature living in Los Angeles. He is the editor of *Bunk Magazine* (http://bunkmagazine.com) and director of communication for the Electronic Literature Organization (http://eliterature.org). His works include Living Will, *A Show of Hands*, and *Marginalia in the Library of Babel*. He was one of ten coauthors of *10 PRINT CHR$(205.5+RND(1)); : GOTO 10* (http://10print.org). He teaches at the University of Southern California, where he directs the humanities and

critical code studies (HaCCS) lab (http://haccslab.com). His complete portfolio is here: http://markcmarino.com.

Published by the Dartmouth College Library.
http://journals.dartmouth.edu/joems/
Article DOI: 10.1349/PS1.1938-6060.A.429