# Group 6. response to Matthew Fuller. "Behind the Blip: Software as Culture" (2003)

Group 6: John Halltiwanger, Sjoerd Tuinema, and Marc Stumpel
Date: 30 November 2009
Sponsors: Jan Simmons and Ed Shanken

In 'Behind the Blip: Software as Culture' (2003) British new media critic Matthew Fuller stresses the need for critical work in the area of software production that goes beyond treating software as merely a functional tool. He calls for an emphasis on software as a cultural phenomena. Fuller is an important key figure in the emerging field of software studies. Next to Manovich and Kittler, Fuller has made early references to the study software as a cultural practice. This allowed other media theorist, critics and philosphers to build upon the concept of software as a culture, which helped to establish the multidisciplenary academic research field of software studies that focuses on the cultural effects of software (systems) by approaching software as a technical artifact, from the perspective of humanities and social sciences.
Fuller first points out that the research that is being done in the area of Human Computer Interface (HCI) is too conventional for the approach that he is proposing; it has the tendency to narrow the scope down to the imposing of functionalist models that cohere as the user. The research area of HCI focuses on the perception of an individual user, with a narrowly applied psychology. Fuller calls for a change in the focus of HCI from static elements in software to the models of involvement processes that software contains. He mentions groupware to explain that individual software users can be involved in wider processes, which complicates the narrow focus of HCI on the individualized user. According to Fuller we should should widen the notion of the processes of the software and their psychology to make an analytical approach possible that recognices its process of becoming.

Subsequently he explains that programmers already have shown that programming can be related to other cultural, social or aesthetic practices. One key example is the programmer Ellen Ullman who writes the following in ‘Close to the Machine’ (1997): 'I'd like to think that computers are neutral, a tool like any other, a hammer that can build a house or smash a skull. But there is something in the system itself, in the formal logic of programs and data, that recreates the world in its own image.' Here software is not just a tool, but ‘composed’ of cultural aspects.

Also, Fuller relates this urge for understanding computer use through computer code back to classical idealistic tendencies. He describes this by naming two contemporary films: Enigma, in which a character describes 'pure mathematics' as 'truth and beauty', and Pi in which the main figure seeks to find a 'numerical meta-reality'. Fuller describes this approach, of absolute beauty to be revealed 'only to those souls which are themselves beautiful', as an nihilistic, unproductive and ignorant idea, since 'numbers do not provide big answers (..) [but] provide access to more figures.'

According to Fuller to undertake theorization of software an understanding of the complex interactions in software processes is required. This would make it possible to study particular objects and their (networked) particularities and conditions; with interfaced computers, networked to other machines, many more levels to research may exist.

While moving towards software studies as critical theory, Fuller wants to clear its reputation by arguing its objectivity of tools, 'which are held back from invading the conceptual domains of software by the myth of its own neutrality as a tool.' Fuller argues that the available tools would be suitable, if they will be finally be accepted. He basically makes two suggestions to make critique software more viable: 'one in terms of scale, the other in terms of activity.' First, Fuller wants research to involve a scope that goes beyond a 'specific instance', mainly because one piece of software could become abundant, though categorial research would instead 'combine shelf-longevity and discourse redeployment', which is what the author seems to aim at.

Also concerned with timescale, Fuller argues it 'should not be determined by corporate release schedule.' In this argument he references Donald Knuth who proposed analyzing the computer processes that occur over a single, particular second. With this approach Fuller questions: 'Why not contaminate this simple telling of the story of what goes on inside a computer with its all-too-cultural equivalent?' This kind of work would supposedly revolve around the use of software that's being documented, although the author is, in our view, not entirely clear about this point. These timescale demarcations would prevent the field to 'end up with a repetition' that's occured in Film Studies. Then the author describes the work of Jakob Nielsen and Donald Norman, in which 'specific problems' are outlined. These theories would revolve around file structures, protocol, sampling algorithms et cetera. These (one layer or one node) insights would be neccesary to understand it's position within 'a wider set of intersecting and multi-scalar formations'. With these 'engaged processes of writing', Fuller seems to again emphasize research on types of technological processes.

In the following section the author describes how critical software analysis could (or should) be received from a philosophical viewpoint. He does this by discussing the 'back to basics manifesto', *What is Philosophy?* by Gilles Deleuze and Felix Guattari. As philosophy should again focus on 'the invention of concepts', the authors dismiss the modern adaptations of the keyterm 'concept', which was widely used by computer science, marketing, design and advertising. Here, Fuller finds the work to be 'fundamentally suspicious' about all electronic media, which in his view 'do participate in 'conceptuality'.' Mainly due to its cultural and cognitive impact, these media 'shape [the world] every time its used.' As Fuller describes the computers (and mainly modern PC's) as assemblages (the aspect Deleuze and Guattari described in 1984), it 'provides an opportunity for critique' and 'theorisation and practice that break free from preformatted uniformity'. This study of metamorphasis should be concerned with what the 'opportunity of further assemblage' could consist of, which has became a key component in modern computing.

In the rest of the article, Fuller differentiates three types of software that he sees as constitutive of software studies. Unfortunately he remains vague on all of them. The first, critical software, is said to use 'the evidence presented by normalized software to construct an arrangement of the objects, protocols, statements, dynamics, sequences of interaction, which allow its conditions of truth to become manifest.' (Fuller 2003: 8). The second type of critical software involves the modification of software in order to disrupt the presumptions of the user in the experience of that software. Here Fuller reference's Jodi's mods of *Quake* and *Wolfenstein* in which the familiar interface elements of those games are deconstructed and re-assembled in non-intuitive ways. The *Quake* mods were released on a CD called *Untitled game*. The project 'tweaks a system yet retains ontological aspects of the system from which it mutated.'[1] (*Untitled game* is also available on the web.[2]) One of the means used to achieve this is the retention of original audio which 'recall indexically in the player's minds eye the original Quake levels and characters. A ghost image of the original flickers behind the alteration, evoked by sound and interface artifacts.'[1] [1]Jodi changes the visuals drastically, allowing for a juxtaposition between audio and visual that both informs and enables the software's critique. Similarly Jodi plays with the expectations of the user to input by altering the experience such that small actions 'become triggers of algorithms that then unfold semi-autonomously.' [1] The commands remain the same (trigger is trigger) yet the result of various inputs unfold in unexpected ways. One such example is the "Spawn" mod which turns the gun into a paintbrush for pixel painting and thus a means of defining an environment in an otherwise featureless black (anti)space.[1] These two axes of diverging expectations, audio-visual and cause-effect, define the boundaries of critical engagement for *Untitled game,* begging the question: what are other potential axes? For example, what methods of critique would best fit a role-playing game such as *Final Fantasy VII?*

The second type of software he describes is social software. Social software is defined (primarily) as 'software built by and for those of us locked out of the narrowly engineered subjectivity of mainstream software.' When Fuller suggest that Free Software can be considered in these terms, it begs the question: what social software exists outside of Free (as in speech) software?[2] What other examples can he name? It seems unlikely that any exist. The guarding statements surrounding free software are inexplicable in light of the absence of any other examples for social software. Furthermore, when Fuller states '[t]he concept of social software too, provides only something small, a little nothing,' it is hard to understand his position. He wishes for a world where software can be studied, yet the software that can be most studied (source is shared openly) is dismissed as a small nothing? Or perhaps he is dismissing his articulation of social software? The issue is further confused by his repeating the common attack that free software has no imagination. His example of KOffice as a clone of corporate thinking, 'dead from the neck up', is not balanced whatsoever by examples such as pyRoom ('distraction free writing').[3] He also misidentifies 'LayTeX' [sic] as an editor, when in fact it represents a radical departure from the corporate WYSIWYG style personified by MS Word. In LaTeX documents are programmed using markup in any editor that supports ASCII output and them compiled into an output format such as DVI, PostScript, or PDF. These sort of omissions and factual transgressions are likely to turn off the technically minded, that is, anyone approaching software studies from the software side rather than the media studies side. This is an important element that should be considered imperative to software studies: if the technical details are not accurate, it is hard to take seriously. In fact, free software has many modes of interacting that are not mirrored in proprietary offerings. Examples such as Emacs and vi, both of which can be controlled exclusively via keyboard, provide experiences cognitively distinct from proprietary offerings. By mirroring the tired critique of the supposed lack of creativity in free software obscures the fact that it is the technical inexperience of such writers that leads them to dismiss open source offerings as uncreative. Perhaps it would be worthwhile to consider the impracticality of, for instance, asking someone only familiar with Word to go to editing LaTeX documents in vi, rather than lazily criticizing the tendency of open source to make as many free alternatives to proprietary software as possible. Other approaches exist, yet the users

---

1. The team attempted to utilize a copy of *Quake* acquired through Steam, but due to copy protection, loading mods is not available. The mods no longer worked with the new open source engine *Dark Places.*
2. Here ignoring the often obtuse debate of free software versus open source and considering them instead as one.

have been shaped by proprietary software to avoid them. In this sense the user model and the user may not be as far apart as Fuller expects. Certainly it implies that just building a new approach to text, for instance, does not mean it will be adopted by significant portions of users, who generally avoid learning curves.

The last type of software, speculative software, 'creates transversal connections between data, machines and networks'. Fuller compares speculative software to fiction's relation to language, that is, it is constantly 'reinventing and expanding upon the capacity of language to create new things.' In that sense, speculative software is "self-conscious" of its existence as software and from that position interrogates its various relationships to the computer, to networks, to users, and to other software. An example of speculative software, developed by Fuller himself, is Web Stalker, 'a technical and aesthetic operation on the HTML stream that at once refines it, produces new methods of use, ignores much of the data linked to or embedded within it, and provides a mechanism through which the deeper structure of the web can be explored and used.' [4] This is a sort of prototype for the kind of software Fuller describes in his section of speculative software, in which the 'blips' that constitute overly simplistic representations (bank account balance) that hide political and social implications and relationships (class heirarchy, capitalist processes) are brought to the forefront. Unfortunately his description remains vague, with no specific means proposed for re-representing these 'blips'. Pulling in political implications in an automated way is problematic, to say the least. Could such software be generated that did not contain a bias towards the authors' own politics? The vagueness of Fuller's description leaves more questions than answers when it comes to speculative software. For instance, can viruses/artificial life be considered speculative software? These are classical examples of "self-conscious" software and are immediately called to mind upon reading such keywords as 'mutation' in relation to software.

In conclusion, we have found Fuller's lack of elaboration to be a constraining factor for the efficacy of this essay. However, the importance of his work in igniting a debate by pulling various strands of already existing research into a new field called 'software studies' should not be underestimated. The largest problems facing the field are a persistent vagueness and a lack of technical accuracy. Fixing these issues is imperative in order for the field to mature in a way that incorporates computer science in its interdisciplinarity. A software studies without such cross-pollination would be an emaciated field indeed.

## References

[1] http://www.opensorcery.net/2reviews.html
[2] http://www.untitled-game.org/download.html
[3]http://pyroom.org/
[4] http://www.chapter.org/2087.html