Matthew Fuller

Endless, multilayered, super-fast and infinitely complex boredom: hooray

What makes software jump? What words, what styles of thought do we need to understand running code and the multi-layered compositions it is part of, and how, if at all, does software establish relations with what might termed freedom? Such questions, of how to act in and understand complex technologies and live situations are not unique to technology, and for a figure by which to understand them, it is often useful to start from the wrong place, not with software, but with a frog. In his book *Lifelines* the biologist Steven Rose describes the way in which a number of his workmates might, whilst sitting at the edge of a pond, compete to describe the leap of a frog. By trade, they are a physiologist, an ethologist, a developmentalist, an evolutionist and a molecular biologist. Each sets their particular disciplinary scale of perception against those of the others. The frog, responding not to the nattering of the knowledge workers but to a snake spotted on a nearby tree, splashes elegantly into the safety of a pond. The representatives of their disciplines each in turn ascribe the "jump event" to: the interaction of nerves, muscles and bones containing and releasing structured patterns of energy and movement; learned or grown behavioural responses; the result of the particular pattern of growth of the organism; the action of an inherited genetic imperative; or the biochemical properties of its muscles.

As the ripples in the pond spread and interact with other movements in the water, Rose's argument is to encourage equally multivalent ways of thinking of a non-reductive biology of life-patterns. Whilst, in his experiments on the physiology of memory, there can be few people in the world who have scissored as many heads off hatchling chicks, Rose's appetite for a wet, complex, living biology is something from which, with all necessary irony, our understanding of software can learn. The trick for biology as a whole, he suggests, is to find a way of engaging both the volition to detail entrained by disciplinary approaches, which are in turn geared to particular constituent scales of reality, those of the gene, the molecule, the organism and so on, whilst at the same time recognizing the radical interweaving of such scales.

If we talk about freedom in relation to software, and after having spat a few times to clear our mouths of a word so enduringly soured as freedom, a word that still however makes our mouths water and tongues wag, perhaps then we can suggest that a similar set of scales might pertain to software. Imagine a group of people watching a computer. One holds that what it does is determined by the hardware, that the mineral architecture of computing is that which sets what is possible. Another looks to the history of languages. They say that software is determined by the kinds of syntaxes buildable, by logical structures that are available in each different environment. The third works through a critique of the political economy of software and suggests that the relations of property embedded in and circulating through it engineer what is possible in software. This person might emphasize the insights of the Free Software movement. Lastly, the fourth figure suggests that software can only be understood by an analysis of the user interface, by an ethnographic querying of the signifying processes of the machine and of its uses. What people do with it is what establishes its quality of freedom. Whilst these figures do not exist in any 'clean' sense, they do represent existing tendencies in the understanding of software and also divisions of labour in its production.

In an aside in a classic essay in the Actor-Network tradition, a current in sociology emphasizing the interaction of elements in socio-technical assemblages, Madeleine Akrich describes the possibilities for developing an analysis of the car. She suggests that such a

study has its natural scale. "Doubtless it could be satisfying to paint on a broad canvas, starting with nuts and bolts, pistons and cranks, cogs and fan belts, and moving on to voting systems, the strategies of large industrial groups, the definition of the family, and the physics of solids ... On what grounds would the analyst stop—apart from the arbitrary one of lassitude? Quite apart from the indefinite amount of time such a study would take, there is also the question as to whether it would be interesting." Mapping the way in which every part of such a complex technical object simultaneously embodies and measures relations amongst 'heterogeneous elements' might be even more draining in the case of software.
Do we need to make this voyage through boredom? What would it involve? A phrase which has passed into the everyday understanding of computing, "the combinatorial explosion", was coined in the 1973 Lighthill report into artificial intelligence. It describes a situation where each element in a logical puzzle needs testing against every other part. As the number of elements increases, the number of such combinations rises exponentially. The problem gets too big. This was seen to set a natural limit to the scale of the then current programme of artificial intelligence and the report was used to legitimize the drastic cutting of the number of centres involved in such research in the UK. Such seemingly natural scales to both understanding and "intelligence" set thresholds of complexity beyond which it is difficult, if not arduously boring to go beyond. The situation is complicated further because, as Edsgar Dijkstra notes, "In computer programming our basic building block, the instruction, takes less than a microsecond, but our program may require hours of computation time. I do not know of any other technology than programming that is invited to cover a grain ratio of 10^{10} or more. The automatic computer, by virtue of its fantastic speed, was the first to provide an environment with enough 'room' for highly hierarchical artifacts. And in this respect, the challenge of the programming task is without precedent."
To map out the elements of software, at any one of the scales embodied in our four watching figures, is therefore complicated by the question of time. Rose sees history, the movement of an organism in time through and as the interplay of these relatively self-determining layers, as a means of understanding biology without falling into the trap of a fetishised, eventually dysfunctional, reductivism. A life is enacted not in conditions of the organism's own making, but each life generates it own combinatorial explosion.
If we were to be unwise, to obstinately disregard Akrich's indubitably correct warning and to try and map out every possible element of the catalytic web of a piece of software, at every one of its scalar articulations, and both at the stretches of speed appropriate to the rate of calculation of its hardware (in its every actual configuration) and for all the social, semiotic and experiential combinations within which it inheres, what might result? One can imagine endless drifts of finely sociological boredom being endured in the study of software. Should we launch global and trans-generational research institutes for the study of every instance of some minor phone-game or for all calculations made in Lotus 1-2-3 or the Sasser virus? Perhaps such studies are already underway, their results remaining as yet stupifyingly incommunicable. What then should we watch out for and what kind of instruments would be used? Will there be some ideal moment of alignment between the four or more scales at which a particular rupture or collapse makes itself manifest?
Perhaps here there is something equivalent to the immense mapping of interactions of the assemblage of a car. Complex artifacts, with multiple and parallel combination of micrometer fine parts moving at speed can suddenly-when particular frequencies are hit, when certain combinations of noise, alignment, heat or other factors coincide-mean a breakdown or worse. A cylinder manufactured to slightly dud specifications hits a pitch of work and it cracks: the physics of solids align abruptly with the history of the car's users. In software, bugs, crashes

and errors particularly in those kinds of software interacting with those versions of the user known as the consumer represent some of the same kinds of interactions.

Multi-scalar alignments can also have different kinds of results. In living systems, such changes in state and in the rate and intensity of interaction between parts can mean the transition to a different metabolic condition, population changes such as speciation or extinction, or the specialization of cells. In terms that are understood as political, and at scales that are both social and inter-personal, combinations of elements may go to such highly visible boiling points. But the way in which they also negotiate, competitively or collaboratively engage to sustain, lock, shift, change or break these inter-relations between elements without crossing such thresholds are also significant.

One such example which, of course involves the all-too-heterogeneous car, is the imperative to imagine feedback between the state of the planet's ecology and the economic and technical forms of the species that currently dominates it. We need a better practical vocabulary than that of reform or revolution to describe the kinds of changes that our current ecological condition requires, but such forms of feedback need not only to be imagined, but to be acted upon and made. Software seems to be an interesting place to begin. Not only does it constitute the domain in which the work of modeling, gathering and number-crunching has been and can be done to intensify time in the right direction, to speed virtual futures up before they happen, to enable us to act on their implications, it also provides part of the domain which can perhaps “out-heterogenise” capitalism and its preferred, ownable, sources of energy.

Traveling through the immense multi-scalar fields of lassitude that are promised for the feckless by Akrich one notices that here and there, there are wrinkles between the scales. One form of knowledge demanding another to makes sense of itself. Amidst the vast terrains of blankness and repetition, the opportunity to find weird little clots of association, parts biting into each other, amidst an immense bland mapping of uncountably similar parts a thousand elements simultaneously freezing, crashing or engorging, generating fissures that open up onto new terrains, new figures of thought, that compel new scales and conjugations of understanding.

There is no “natural scale” to software. Each of the four, inevitably more, figures must collaborate, work on itself and in liaison with the others, and recognize its occurrence within live history. The challenge that such an understanding of software makes is multifarious, and simply linking layers is not enough. In certain kinds of metabolic web, previously discrete elements network together so strongly that they crystallize, effectively becoming one component. Multidisciplinary models of work, linking these scales and connecting hardware designers, programmers, lawyers, economists, user interface designers, sociologists and marketers certainly exist; they call them corporations. But even in and between such entities, styles of knowledge and forms of work are hard to make monolithic. The scalar solidarity of programmers within Free/Libre and Open Source Software movement has spelled out one set of ways to speak of freedom in relation to software. At the same time, it alerts us to other uses of the soured word: the search for crash-proof killing machines has spurred the partial incorporation of Linux development by the US military; but who better, one argument goes, to subsidize the communism of ideas? The four scales of software sketched above suggest that freedom must be made at other levels too.

In our travels through the endless fields of boredom how can we tell whether we are at the edge of some particularly interesting fold or intensification? How can we smell out a potential alignment of elements within scales that promise a potential figure of freedom? What instruments exist to decipher whether or not we are at the edge of a moment at which everything, at every scale, becomes crystalline and rancidly clear, or, as figured in the debates about the militarization of Linux, whether domination is possibly over-stretching itself, becomes its other in order to race fast enough against history to keep everything the same? A similar question could be posed about moves to apply unsuitable patent laws to software. Is there an algorithm by which one might calculate the potential interactions within these four scales of software?

First, such an algorithm would have to constitute itself through paradox, involve a recursive disassembly and reconstitution of the notion of freedom itself, and an equal dose of laughter at the notion of transcendental rules. Its tongue-in-cheek rules of thumb might draw upon "Do What You Will", Rabelais' minimal rule-set for the utopia Thélème, or the libertarian's handy slide-rule "It is Forbidden to Forbid", and of course there are countless others. Yet maxims are not enough. Both freedom and software require a certain lightness coupled with an obdurate recognition that it is possible to make something happen, that it is possible to make leaps, at any scale. Such algorithms would be capable of working in, amongst, and beyond, the four scales sketched earlier, sucking their constituent elements into unimagined conjunctions with elements traversing other realities into quick condensations of processing. Irreducible to themselves, and thus also impossible to configure as software only, perhaps those affirmative and convulsive elements and conjunctions can be found busying themselves or waiting in the strange knits between layers of reality. Working relations of capacities and energies over time, software's structural and constitutive interplay of constraints, affordances and invention, mean that the figures of freedom we will require in order to navigate boredom will have to be enacted as much as encoded.

Akrich, Madeleine, "The De-scription of Technical Objects", in: Bijker, Wiebe; Law John (eds.), *Shaping Technology Building Society.* MIT Press, 1992

Dijkstra, Edsger Wybe, *A Discipline of Programming,* Prentice-Hall, 1976

Lighthill, James. "Artificial Intelligence: A General Survey", in: *Artificial Intelligence: a paper symposium.* Science Research Council, 1973

Rose, Steven, *Lifelines, biology beyond determinism,* Oxford University Press, 1997