Fun and Software

Fun and Software

Exploring Pleasure, Paradox and Pain in Computing

Edited by Olga Goriunova

B L O O M S B U R Y
NEW YORK • LONDON • NEW DELHI • SYDNEY

Bloomsbury Academic
An imprint of Bloomsbury Publishing Inc

1385 Broadway
New York
NY 10018
USA

50 Bedford Square
London
WC1B 3DP
UK

www.bloomsbury.com

Bloomsbury is a registered trade mark of Bloomsbury Publishing Plc

First published 2014

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the Library of Congress

ISBN: HB: 978-1-6235-6094-2
ePub: 978-1-6235-6756-9
ePDF: 978-1-6235-6887-0

Typeset by Fakenham Prepress Solutions, Fakenham, Norfolk NR21 8NN
Printed and bound in the United States of America

Contents

Acknowledgements

This book grew out of an eponymous travelling exhibition that took place at Arnolfini, Bristol, UK and MU, Eindhoven, the Netherlands in autumn 2010 – winter 2011. The exhibition had a slightly different composition in each country and was funded, respectively, by Arnolfini and MU (from the Ministry of Education, Culture, Science (OCW), City of Eindhoven and the Province of North-Brabant), Baltan Laboratories, SNS Reaal, VSBFonds and Netherlands Media Art Institute. Many thanks to all the artists who took part.

I would have never been able to curate these shows were it not for Geoff Cox, who initially suggested doing something at Arnolfini, and Annet Dekker and Annette Wolfsberger (at aaaanet.net) who put so much energy, dedication and fundraising and organizing efforts into the show that they made it happen. Many thanks to them and to the directors and staff at Arnolfini, MU and Baltan.

In November 2010, Baltan Laboratories hosted and, together with MU and STRP festival, funded a one-day conference, where many authors of this volume came together and tested their ideas of fun on each other. This helped shape the book.

Having taken part in the software art scene, the work of software artists and close colleagues of mine from the times of Runme.org has influenced this project: thank you to Amy Alexander, Alex McLean, Alexei Shulgin and the many other people who made software art and digital art such exciting practices. Readme festivals contributed to the aesthetic exploration of computation, and the project *Textension* by Joshua Nimoy (1999), which won one of the prizes at the first Readme 2002 in Moscow, served as an inspiration for the cover of this book.

Matthew Fuller, Andrew Goffey, Graham Harwood and Adrian Mackenzie at Computational Culture journal help make the field of software studies vivid and open. Thanks to them for inspiration and dialogue.

I received generous comments and suggestions on this book from Celia Lury, Luciana Parisi, Nathaniel Tkacz and numerous now unnamable people that responded wherever I spoke about it. Thanks to all of them.

My mother's admiration of Knuth imbued me with an early sensibility for fun in programming. This is for Nikolai, who has intrinsic understanding that computation is fun.

Introduction

Olga Goriunova

Darling sweetheart

You are my avid fellow feeling. My affection curiously clings to your passionate wish. My liking yearns for your heart. You are my wistful sympathy: my tender liking.

Yours beautifully

M. U. C.[1]

In 1949, when the Ferranti Mark 1, one of the first computers, was manufactured for Manchester University, Christopher Strachey was a school teacher. He received the computer's manual from his former university friend Alan Turing, and began the activity for which he was later proclaimed one of the originators: programming. Strachey created an algorithm that would enable the computer to play draughts (checkers): he wrote the program on paper and was given access to MUC (Manchester University Computer) for one night only, to load the program onto the machine and try running it. In the morning, when staff returned, MUC was playing draughts.[2]

> Strachey sent his programme for punching beforehand. The programme was about 20 pages long (over a thousand instructions), and the naivety of a first-time user attempting a programme of such length caused not a little amusement among the programmers in the laboratory. Anyway, the day came and Strachey loaded his programme into the Mark I. After a couple of errors were fixed, the programme ran straight through and finished by playing 'God Save the King' on the 'hooter' (loudspeaker). On that day Strachey acquired a formidable reputation as a programmer that he never lost.[3]

On the basis of this project, he acquired a job in the Computer Department of Manchester University, where he worked alongside Alan Turing, and proceeded, in 1952, to write the Love Letter Generator, an ephemeral, even flippant, poetic program that can be retrospectively seen as one of the first text-generating algorithms.[4]

When David Link, an artist who reconstructed the Love Letter Generator in an eponymous art project, was giving a talk to the Computer Conservation Society in London about the hurdles of the process, including trying to read the program in faded pencil on the few papers archived in Oxford's Bodleian Library and locating the remaining original switches from the operation console, a member of the audience stood up. This gentleman said that he had been a student at Manchester University in the period discussed and remembered seeing nonsensical love-letters printed out on paper stuck to the walls. He went on to say that he and fellow students would never have thought that their 'Gods', such as Turing and Strachey, might have engaged in such a nonsensical activity.

This example provides an easy route into one scale of the concept of fun that this book proposes. The fun that most readily comes to the fore here is that which we are most aware of, the fun of 'geekiness', wit, the humour and extravagance of mathematical geniuses and their material processes of thinking and making. Humour comes before pragmatic application and, as things are being developed and tried out, it forms a part and serves as material of thought. Such fun, programmers' humour, can be seen as related to the culture of the humour of the exact sciences, where, for instance, in the Russian and Soviet reverence for mathematics and physics, books such as *Physicists Joke* became national bestsellers in the 1960s.[5] Humour here is indicative of the pleasure of breakthrough, of thought experimentation, of unconstrained and purposeless but intense and curious and materially specific enquiry.[6] Many a Soviet child, including me, from as early as they could remember, heard tales about their parents' attempts to pass Mathematical Analysis and had their bookshelves lined with the volumes of *Mathematics Can Be Fun* and *Physics for Entertainment*. Yakov Perelman, a great Russian popularizer of the sciences, published the latter in 1913. Perhaps the first thing to do to advance the idea of fun discussed here is to imagine *that* fun of physics, algebra, geometry, of a pre-World-Wars era. Fun is a problematic word, requiring qualification or defence as soon as it is introduced, as it readily connects both to the complex machinery of exploitation through creative enjoyment and to stupefaction through consumption. But it is also a word that cannot be avoided, which unites pleasure and wit with affective engagement, conceptual, social and cultural practices and vivid, disturbing and inventive modes of thinking and making.

When manifest in particular hacker performances and objects, fun may come in the guise of wit, becoming a game of intellect where it can be operative as an 'intelligence test'. The ways in which humour and technical ability are deeply interconnected in hacker cultures can be interpreted as almost a structural similarity between a mode of thought imbued with a good sense of wit and technical or mathematical proficiency. Expressions of wit are judged on the economic scale: Freud in particular used economic thinking to judge wit, where the most successful joke relies on scarce means to achieve the richest outcome. There is a direct connection here to 'coding elegance'[7] and 'the art of computer programming'.[8] Wit can become a skill, a training program and a grading tool to teach, learn, progress in and evaluate hacking.[9]

It is important that clever tricks and elegance are not the same: Dijkstra especially warned against a culture of 'clever tricks', asking for discipline and humbleness.[10] Fun does not equate to intellectual superiority and perfection. Fun can be modest, misleading, a noisy chimera. Here, there is a sense in which software ephemera, digital folklore – as a variety of cultures that use, adapt and produce software that make and 'change' sense in relation to labour conditions, states of work, certain aesthetic normalities, software operations and allowances – always stay minor. At the same time, aspects of such cultures of using and misusing conventional software, along with digital art and software art, have made their way into the world of apps and handheld devices, where they are often severed from their idiosyncratic operationality, and of the ways in which the formation of a shared subject always hung in between the dysfunctional desktop computer and its tense user.

There is an alignment of the ways in which exploratory material imagination proceeds in geek cultures and art work, garage cultures[11] and in the aesthetic mode of work. There is a certain artistic nature to geekiness, the way in which objects and processes, even artistically thrilling projects, are produced within systems of coordinates which are often not interested in art at all. Fun here manifests in a certain proximity of ways of imagining, making and thinking, and of enunciating things with a dose of audacity. There is a metaphysical braveness that is traditionally discussed in relation to art, in critical accounts of creativity or in the feminist distrust of 'genius', that has contrarily also recently been hungrily grabbed at by the managerial discourses of more efficient value-extraction. What this book suggests is to reclaim and think such fun in relation to computing in all its specificity, thus opening it up to radically varied interdisciplinary criticism and thinking.

Images of enquiry into fun

The master algorithm designer Donald Knuth speaks about his attempts to answer 'puzzling questions that have no immediately obvious relation to any practical applications of mathematics or of computer science'.[12] The non-obvious and not immediately applicable, non-instrumental or straightforwardly confounding aspects of computing, such as Turing's deliberately erroneous manuals,[13] or Knuth's 'monstrous' algorithms, shed further light on the kind of fun that this volume seeks out. Fun in computing as a mode of thinking, making, experiencing is about a mode of reasoning, and as such it convolutes the question of rationalism, the qualities of logical reason, the autonomy of computation and relates to the sensibilities and creative drives of computing. Fun is ambiguous about the modes of knowledge and their distributed nature. Fun, within and with logical reason, relates to paradox, the incomputable; it produces sense differently, complementing formal logic while being generated by it. Exceeding the structures of the production of sense, fun is an excess, linking to instability and ambiguity both outside and within the computational, running diagonally across them.

The problem of reason manifests where there is incongruity, growing out-of-scale data quantities, material affections. Such conditions may be found within formal logic, in paradox and the incomputable, and the coupling with real-world processes, and transversally, across the political, experiential registers, and in the computational feeding on the computational. Across the scales at which computation is distributed across society and culture, such modes of knowledge and practice necessarily draw upon and into itself the living material cultures, the bodily, affective practices of programmers and users, social circulatory diagonals, ontological and individuating processes, all feasting upon the centrality that calculative infrastructures attain today. In such a condition, the question of fun is one of metaphysics, enquiring how certain ontological processes are being constructed and enter into conditioning relations with each other. Fun cuts across multiple scales and connects to what drives desire and creates the conditions of the mode of work and life.

Computational culture has fun as its medium.[14] Such fun traverses the formal logic governing computing, and the larger systems of thought and practice within which it is nested, it cuts through hackers' pleasure, pain and computing's social lives and it is not by chance that it also becomes captured in exploitative devices employed by late capitalism to extract maximal value from subjects labouring in computational infrastructures.[15] Capitalism constantly shifts its boundaries to include further lines of escape: 'its lines of escape are … the very conditions for its operation'.[16] Fun that includes pleasure, humour and play, but also paradox, incongruity, ambivalence and pain, and that takes part in the construction, maintenance and use of those computational machines through which creative populations produce value, is part of the spirit of today that we have to problematize.

It is easy to criticize a neoliberal impetus to be creative and unpaid and lead a meaningful life by purchasing unique experiences to report on websites or by obeying the prompts to pleasurably play (with funware, game-like mechanics on websites) when shopping, filling out forms, etc. What is significant is that there is a larger fun that partakes in going beyond oneself, thinking from bare, breaking outside within repetition, along with the luxury of excess[17] and the vertigo of play,[18] disturbance, despair and purposelessness. Fun as a jaw-dropping, dopamine-releasing exercise, the fun of learning and being happy, as captured by neuro-imaging, continuously colonized and monopolized is intermingled with fun enjoyed in the form of the process of actual occasions (algorithmic processes) in their determinacy via indeterminacy (Parisi, Fazi), the fun whose possibility arises out of the processuality of computation (Fuller) and the fun rooted in the inherent excess of real-world functionality bringing the affective into computing or the structures of code and computing practices functioning aesthetically (Goffey), to name just a few approaches. Not only the unserious or the defiance of the self-disciplinary order of rationality can be called fun or be captured (as discussed above), fun in computational practices helps us consider computing as affective, ambiguous, autonomous – to just begin with a few qualities whose linguistic codification starts with an A.

This book proposes that there is a distinctiveness of fun in relation to, broadly speaking, computing and computation. Fun here is related to abstraction and notation, formal logic and experience, singularity and repetition, contingency and practice, incomputability and paradox, to the question of where computational processes start and end and which assemblages they construct and are part of, to the subjects and experience of and by computing, to how algorithms model and are modelled by the world and how they couple with other processes, within and outside computation, to the cultures and aesthetics of software, to its building upon itself, to its politics, desires and legal projects.

About this book

The starting point of *Fun and Software* adheres to the proposition of software studies: today, computational structures undergird, enact and take part in actualizing nearly all societal formations and fields of action and hence it is urgent that we investigate software critically.[19] With the promise of big data, with mass surveillance and individualized measurement across scales, new regimes of governmentality use calculative infrastructures and give preference to new forms of decision-making predicated upon the analysis of data. Political and social formations and belief are produced, maintained and governed through calculative devices and models, which move on to manifest as intimate companions rather than bureaucratic monsters.[20] There is significant work focusing on the unpacking of the informationalization of finance.[21] The cultural, aesthetic and existential forms of production are not untouched by computation: to exist, relate, enquire, think, take pleasure and suffer is structurally coupled with computational machines, mathematical models, rankings, networks, database management systems, code, algorithms and formal logic. The production of subject and subjectivity is not only a process in which computation takes part; it also becomes unclear what kind of subjectivity and subjecthood is produced and whether the position of the subject itself is eroded enough to be contested by other forms of living, reasoning and processing. Subjectivity also links very closely to objectivity, between which fun becomes a curious method of connection.

Computation here must not be seen as an opponent infiltrating supposedly pure prior states of being. As computation takes part in becomings, we must look for the ontological specificity of the computational. To do so, the forces of the humanities and arts, philosophy and cultural theory must be called upon. The crisis of humanities demonstrates a mismatch between the necessities and the capabilities of today. Substantial currents in the humanities and social sciences still often regard computing as both a black-box and an insurmountable fortress guarding the gates of formal reasoning. These disciplines and the arts are also under attack for their lack of supposed utility or financially incentivized functionalism, for the genesis of which computational infrastructures, as they have been recruited by cognitive capitalism, can partially be blamed. However, computing is not exhausted within the framework in which neoliberalism puts it to use. What this book aims to do is to create conditions in which computing can be seen as an excessive and intensive scale and order, which can and do take part in forming and acting in transversal ensembles that are more than those of the neoliberal machinery, administrative applicability or calculative completeness and functionalism.

This book draws programming, software and computation back into social, cultural, philosophical investigations. What we attempt to do is to foster cross-disciplinary forms of attention to software and computational processes that look at how software and computing are imagined, thought, made and experienced. The volume also focuses on how computing takes place, makes sense, recruits and extends, folds upon itself, exists. This book is an attempt to expand computing, to inhabit it through becoming open to the arts and humanities, social sciences and less disciplinarily defined modes of enquiry.

However improbable it might sound for today's all encompassing dullness of forms, databases, schedules and processors, 'fun' has informed and guided the development of computing from its very inception. In a book on the history and secret of scent, Luca Turin writes that smell, a sensory dimension that most people experience daily, is exclusively described and researched either in relation to war (gas attacks) or commercial use (perfumery).[22] A certain similarity can be found here to the ways in which software and computation are researched. The history of computers is largely seen as emerging out of military imperatives, cold war, threat, violence and death. And in many ways it does emerge from war, as do many other things, including me or Donna Haraway, which she describes as following: 'I have a body and mind as much constructed by the Post Second-World-War arms race and cold war [as women's movements]'.[23] Paul Edwards, in *The Closed World*, suggests that there are two versions of the history of computing: one that is the history of logic and symbols and the other that is a history of computers as information-processing machines.[24] Michael Castelle argues for a history of computing that takes into account administration, statistics and bureaucracy.[25] Military, industrial, logical histories, however accurate they are, contribute to constructing the illusion of pragmatism and necessarily omit other stories affined in their non-teleological thrill.

The concept of fun helps bring together the ways of thinking and experiencing computing and the modes in which computation acts, which are usually presented as marginal and seen as unrelated to its ostensibly more serious or useful dimensions. In *Fun and Software*, software is taken out of its exclusive corner to be looked at as a field of thought and practice informed and guided by paradox, ambiguity, affect, which produces agents, subjects and practices of new ontological orders. *Fun and Software* suggests that fun is a means of production and multiplication of sense in and of software – something that is only supposed to have one, useful meaning, or to be solely serious, precise and spotlessly operative, whereby fun becomes both a concept and a method, an 'onto-epistemological' enactive device.

Fun as desire emphasizes the extent to which the investment and coupling of desiring-production with the computational structures is central to today's society, its forms of expansion and repression and is therefore politically crucial. As computation has found a way to everywhere in society, the texture of such desiring necessarily outstrips and feeds the formal logic of computing while being organized by and into it. Emphasizing and exploring the multiple scales of fun in computing creates conditions in which forms of critical, political and collective expression of desiring fun can be seen and given voice.

In reclaiming fun, and especially fun in computing, there is a chance to imagine that the world does not only run following a deadly logic of economic, military and other forms of violence, but has trajectories that follow bizarre, complex, joyful and open vectors and enquiring passions which are also both ontologically and epistemologically embedded or latent within the computational. It is when fun is had that the world is invented. Without imagining this, futures we would like to inhabit cannot be enacted. In some way, the aim of this book is to carve a breathing space for subjects of different orders and kinds. 'The Gods are dead, but they died from laughing', says Deleuze for Nietzsche.[26] The fun to be explored in this book is the noble, audacious enquiry that imagines the world in ways that deserve recouping.

Complicating fun

Peter Berger notes that modernity has a comic sensibility: 'the turn in philosophy from a negative to a positive assessment of the comic only came in modern times', with Erasmus's *In Praise of Folly* 'serving as a bridge between the comic culture of the Middle Ages and modern understanding of the comic'.[27] Comic becomes a signal of transcendence. In *The Gay Science*, Nietzsche talks about laughter and the ability to laugh, withstanding meaninglessness, as an ability to transcend 'values' that assign reason to life. Such laughter does not have any purpose (such as development); it signifies a new era of a 'gay science' taking place after the tragic is over.[28] Nietzsche proclaims that the tyranny of science and truth will raise the value of falsehood, and the tyranny of prudence will force into being new kinds of nobleness, 'nobleness to be capable of follies'. In such laughter the value of folly is raised in a way that enables reason to be multiplied and changed. Nietzschean laughter here is about a pure difference with the energy to trigger change.

The fun this collection explores is such a fun that breaks and makes meanings, passions and actions. Not, or not only, a Bergsonian confirmation of value, or a Freudian resolution of conflict or an estrangement device, fun does not operate through synthesis or catharsis. Fun as an operational mode is more intense than the displacement traditionally ascribed to humour (though fun is not afraid of humour and may include it as its integral part): whereas humour takes on estrangement, incongruity or relief, an abstract relationship to reality that allows for re-evaluation of the latter, fun has the potential to disturb the status quo and operate at a more profound level, at a tangent. Fun, akin to Nietzschean laughter, includes disbelief, undoing;[29] 'to laugh means to love mischief, but with a good conscience'.[30] This mischief is the transversal paradigm of cutting across and assembling in incongruously radiant ways.

Fun shares affinities with humour, play, wit and laughter, which all have a relationship with the true through existential conditions. Kierkegaard in particular regards humour and absurdity as a separate existential stage, the last state before faith, a kind of leap[31] – and it is interesting in this respect that when Haraway introduces 'A Cyborg Manifesto', with a discussion of irony, she also deems it a blasphemy.

Fun can be seen as related to a specific history of thinking laughter, explored by thinkers such as Nietzsche and Bataille, as an ability and a condition of staring at the limit of the possible, beyond the accepted or agreed and without an immediate resolution. Such a path is particular in its inclusiveness of a very profound kind of joy and passionate sadness that are emotional as much as they are active, powerful and conceptual.

The above-mentioned Edsger Dijkstra noted that 'a clever programmer [is always on a task to]… squeeze the impossible', but the impossible is not about better efficiency or clever tricks *per se*. The task of such an understanding of fun is not simply to equate powerful abstract thought, ways of thinking and doing software with 'testing the impossible', but to understand and, most importantly, recognize and construct such histories, working through the being and practices of computation, modes of thinking, hacker ethics, cultures related to software, aesthetic work which themselves have a unique and powerful ontological force. Such a force is the idiosyncratic vector of fun along which human-technical ensembles are brought to life and develop, which works across scales, subjects and materials, without losing the differentiating quality that is particular to it.

There is a particular fun that finds its fascination in logic. The acquisition of language, notation and the establishment of the practice of programming include fun as a mode of production that oscillates between abstraction, normativity and paradox or the absurd. Computation contains the incomputable. Paradox and incompatibility exist as forms of friction between computing and mathematics, and between abstraction and the real, and found one of the forms of computational experience where fun resides. Ambiguity, thus, is an essential mode of the aesthetic of computation, connecting to the processuality and collective physicality of computing.

On the other hand, the extreme limit of the 'possible' requires insane courage, for which, to paraphrase Bataille, madness can be the repayment. Fun is a process, the price of which could be illness, and folly. There is a tragic fun, the dark side of fun as formally aligned with anguish. 'Humour has the same formal structure as depression',[32] where the humour is about suffering, sublimity, inauthenticity, impotence or violent ecstasy.

Fun can itself become a capability of folly, can make folly manifest or be a mode of dealing with it. The ways in which fun as desire can and does couple with the organization of power, social, technical and material bodily machines, can imply a subtractive logic, can make weak and diminish. Indeed that is the common knowledge within the cultures of striving for, building, embodying and carrying out the material encounters and interminglings with computational processes, and is often described as such in critiques of computational capitalism.[33] Fun is related to intensification, obsession, dreams, exploitation and abuse. Fun can be manic, tragic and evil. Malware and evil media,[34] psychopathic and neurotic lives and the schizophrenia of capitalism,[35] also gild and frame the complexity of fun as a manner of action in the world.

It is important to remember that fun is not an abstract principle; it is formed by material relations and actions. Fun is a processual unfolding found in movement – distributed and machinic – it is in amalgamations, splits, working through bodies, objects, structures, data and actions. Fun constantly negotiates between subjectivities and collectivities; it is a technical individuating force. The processual dynamics, the mode of performance of fun can be expressed as data and worked upon by algorithmic structures or can be enunciated in computational forms inhabited socially and collectively. As software enunciates, takes part in affection, becomes a rhethorical, legal and ethical project, fun performs and circulates, governing specific forms of spontaneity, liveliness and sociality and organizing time–space.

Overall, fun is of different scales and types: here in this book it relates to forms of reasoning and the becoming of computing; it is experiential, affective and collective; it is a desiring process and a sensibility, a mode of thinking and of working; a horizon, an idea, a passion and an action. Fun should not be understood correctly or in one way: its strength is in its multiplicity.

Summaries and relations between chapters

As becomes clear when reading this volume, every author approaches fun in a distinct way. As this is a collective project, the chapters also construct fun as a multiplicity, with several themes complementarily explored across chapters.

Recurringly, authors address the question of whether it is possible to move away from the human subject as the one experiencing fun or formulating a logical procedure and look for fun that belongs to programming structures, practices and real-world contexts, where its subject is distributed. Addressing the processual and contextual character of computing, it becomes possible to contemplate experience, ambiguity and incongruity as residencies of fun. This draws attention to the experience that is locatable in the body of the programmer, the affects and passions of the practices of programming. However, experience is also something more widely constructed by computation and constructive of computation and arises from larger assemblages not necessarily fully related to the human arranged as the proper subject. Here, the question of the subject is shifted to that of operation, and whereas reason and formal reasoning are still linked to the subject, the nature of logical reason can also be seen as enjoying some autonomy, while computing becomes intertwined with design, art, passion for data and playfulness, among other things.

Constructing the subject of the programmer, user or player in the bodily passions of achieving singularity through repetition, the acknowledgement of bodily enjoyment and pain, psychic energy, the practice of programming raises the question of power. Negotiating political relations between the subject and the object, enunciation and exploitation, production and diminution, fun is seen by a number of authors here in relation to multiple vectors and coalescences of power. The themes of code and action, both bodily and symbolic, in their political potentiality also cut across chapters. As an example of shifting value from code to practice and context, free software is often discussed as well as the enunciative qualities of code.

The role of paradoxes, the incomputable, the inclusion of ever-growing real-world data producing interruption and introducing unpredictability to the algorithmic processes form an important line of thought fleshed out by many of the chapters. Necessarily rather than predictably, humour, play, jokes and games are also finely attended to. Many chapters also stress the importance of considering experience, context, collectivity, bodily material contexts and practices of programming and surpluses of formal system.

Chapter 1 is a programmatic chapter; it asks for how we can step beyond 'the formal-logical categories of mechanized mathematics' and draw upon computing's connections to art, virtuosity, free speech and play. While criticizing the efficiency and utilitarianism of a strand of 'funology' in human–computer interaction (HCI) (linking to the history presented by Kaltenbacher), Goffey questions the assumptions of instrumentality, rationalism and utilitarianism rife in the epistemology of techno-science. Critically re-reading Marvin Minsky's work, Goffey looks for a connection between fun, programming practices and the structures of software. There is a 'margin of decoding', an inherent excess of real-world functionality that brings the affective and the material back into computing. Elements and structures of computer code and coding practices in relation to the real world also function aesthetically, as exemplified most clearly in the messy enunciative qualities of software.

Chapter 2 takes the reader on a journey through the philosophical, mathematical, logical, notational, literary and aesthetic genealogies of computing via some of the key figures in the establishment of the discipline of programming: Dijkstra, Knuth and Kaufman. Tracing the shaping of programming paradigms, Yuill moves through the liberal philosophy of Mill and Bentham (Knuth's inspiration) and Wittgenstein (Dijkstra's inspiration), Knuth's jokes and *A Fortran Coloring* book, Turing, Bourbaki and Rotman, FORTRAN and ALGOL. The human becoming machinic and vice versa are traced alongside their embodiment in programming paradigms via the pleasure of humour or the monstrous, among other registers (symbolic, notational, rhethorical, educational). Yuill explores deep connections between mathematics, programming and aesthetics, questioning computing's ontology and the specificity of human-machinic assemblages of computationality, that, first of all, laughs.

Chapter 3 addresses fun through the concepts of experience in the culture of mathematics and the machinic fun of certain computer games. For Fuller, paradox and ambiguity arise from within computational forms as they move between scales and contexts. Whether fun arises in experience, the texture of its processuality and the locus of such experience and the assemblage producing it are the questions that the chapter pursues. Reading Turing with Brouwer, Fuller suggests that calculation is always a relational process, occurring in time. The processuality of computation exposes it to the entry of experience, reflexivity, time and things formally outside mathematical logic which become part of its living assemblage. Drawing upon the examples of MMORPGs, the art project Human Cellular Automata and the celebrated sandbox game *Minecraft*, Fuller argues that paradox and ambiguity are aesthetic modes through which computation itself becomes a form of distributed and machinic fun that is ambiguous, preposterous, tautological, perverse or delightful.

Chapter 4 delves into the possibilities of autonomy for mechanized thought (as found in computation). Parisi and Fazi ask whether algorithms can be said to enjoy themselves while processing data, where fun is not about breaking free, but about the fulfilment of procedures and thus the achievement of autonomy. Grounding their work in Whitehead, the authors look at algorithms as actual occasions. Not only a discrete set following a procedure until completion, algorithms also relate to the indeterminacy of eternal objects through incomputability, where 'completion cannot always be attained'. Drawing on Turing's formalization of the incomputable and Chaitin's description of algorithmic randomness, Fazi and Parisi establish that, for the completion of the algorithm to be achieved, the procedure must rely on the incomputable being intrinsic to computation itself, where fun is the unfolding of determinacy via indeterminacy.

Chapter 5 focuses on the ways in which programmers embody data structures as the latter channel, fold, slice and sort subjectivities as much as other kinds of realities. Relying on the psychoanalytical literature on aggression and eroticization, Mackenzie looks for 'energized forms of excitement', where the ways in which psychic structures complexly interrelate with desire may be thought of in relation to software and data through the example of the statistics-oriented programming language R. The data structures and statistically informed models that R extends into desires, actions and beliefs transform data, simultaneously seeking to control it while making it more volatile. The tensions between aggression and free association multiply with a modelling of modelling, where a desire of data munging is processed to hallucinate about data productivity. Coding and populations are brought together, posing the question of whether 'bodies of code themselves can become population aggregates and micro-multiplicities'.

Chapter 6 explores the practice of programming and offers an analysis of its truly Spinozist passions. Caught between ecstatic singularity and conformity to convention, particularity and abstraction, programming practice is a passionate, painful bodily experience. The striving for concise expression and logical singularity is coupled with the recursive: the craft of programming is grounded in repetition. Programming frameworks are 'condensations of practice' and the establishment of the pattern is embodied in the practising programmer. Arguing against the idea of programming as disembodied, Murtaugh discusses extreme programming (here his chapter links to Chapter 8) and the dark, unspecified and absurd that join forces with the logical. Transposing attention from code to the practice and context makes computing not a precise and singularized logical abstraction, but something messy and perpetually incomplete.

Chapter 7 starts with an overview and interpretation of a set of seminal literature on humour, jokes and laughter including Freud, Deleuze and Virno. Evaluating the innovative roles of humour against its instrumentalization, Cox and McLean analyse humour as deviation from the normativity of coding practices and focus on the social dimensions of jokes in code. Diversion and semantic ambiguity, breaking the rules of coding practices and norms of source code interpretation, jokes, which are code and action, can release productive desiring-machines. Multilevel interpretative work on source code, where both software and human are interpreters, produces software development with a body, and breaking the rules allows political potential to intensify. Cox and McLean discuss the specificity of such geek humour through the analysis of the brainfuck language and a series of art projects, to name just a few examples.

In Chapter 8 Chun and Lison propose a conception of fun as a dialectical battlefield, between subject and object, bodily enjoyment and obsession, 'good' and 'bad' programming, where the question of the boundary between these states creates the ethics of fun in relation to power. For them, fun manifests through the figure of the user, whether such a user is a gamer or a programmer (a user of 'structures for organizing complexity'), who in turn becomes used by affective structures of the conditions of immaterial labour. For Chun and Lison, the bodily passions of pleasure and pain, related to the uncanny striving for power, empowerment and disempowerment produce programmers, and the chaotic surplus of fun may destroy them. Freedom then is a question of becoming disempowered in a good way, when programming is not *logos*. Chun and Lison draw upon the history of software and the rise of structured programming, and use a rich range of examples, varying from the NetHack game and rave parties to dot-com-era's office design and the Obfuscated C Code Contest.

Play as a multivectorial investigative activity relying on the engagement of the body in relation to sense-making and work is explored by Andersen in Chapter 9. The frivolity of play is mined by an analysis inspired by Deleuze, with the concepts of demonstration and manifestation in humour, which disrupt signification. Such new logics of sensation emerge in relation to context, to those who play and the absence of their bodies. Andersen explores the game *Spacewar!* as a response to cybernetics and the human mediation of control in cybernetic systems. Here, the game and play are a pop cultural parody of SAGE and everyday work, but a parody that draws upon the workspace and its leading conceptual frameworks. The post-war subject, the computer scientist experiences the shifting processes of the production and maintenance of subjectivity and reasoning, and here game production challenges the emerging status quo through demonstration and manifestation, through aesthetic practice.

In Chapter 10 Kaltenbacher looks for fun in the history and practice of HCI. She accounts for how the cybernetic origin of HCI, exemplified in the production of the human subject in terms of its correspondence to rationalism, automation and prediction, gives way to the understanding of (user) experience and emotion. Usability, user-friendliness and emotional design supersede each other, and new concepts come on stage: pleasure, surprise and enjoyment. Fun here is rooted in the complexity of emotions and the ways in which emotions form part of cognition, in learning, play and engagement. Kaltenbacher uses the concept of flow as intensely engaged experience to arrive at a staging of fun that includes experiential learning, experimentation and play. Pleasure and fun here derive from creativity and collaborative experience, and also active and involved participation, which is more pleasurable the more it gains depth, something that Kaltenbacher articulates through situated cognition, new media literacy and implicit learning.

In Chapter 11, Dekker focuses on art and an aesthetic of fun. She enquires into fun as an aesthetic method through the analysis of works by artists, such as: John Körmeling, Aram Bartholl, JODI, Cory Arcangel, Evan Roth (F.A.T. Lab) and Constant Dullaart. Exploring the plasticism of code to reveal the processual performance of coding, the works instruct the fusion of aesthetics and politics. Here fun aligns with upsets to the materiality of computational structures. The methods and material of releasing and documenting the making and the structure of the art work and the ways in which it gauges its functionality are decided differently among those artists. Manual gesture, pleasure of working through the computational material link to the questions of openness, whether legal or viral. A collective, iterative production of fun is part of its aesthetic methodology and nonsense enters the discussion of fun through a light touch, rich in idiotic, ambiguous gestures. The banal and rubbish, as well as the coarse, are mobilized as responses and affects intensify and circulate across the scales of art's existence.

The concluding chapter reflects on fun in the material and constructive imagination of the future as enacted by the avant-garde movements. This take on fun (which is, to an extent, purist) links art, avant-gardes, digital and software art, time and computation. The recursive avant-garde becomes looped and delayed in a perpetual present where the projected future of past avant-gardes becomes our only available future, in the form of the future-in-the-past. Such morphing of times is produced not least by the computational mediation of time, which is analysed through time-based effects in digital signal processing. Focusing on sonic realities allows for an enquiry into the structure of the digital delay line in the production of the off-now. The scintillating unattainability of the crafting of the future performed by Russian sound technology pioneers partially shares its sensibility with the net art and software art of the 1990s and early 2000s, and it is the call for the multiplicitous futures constructed through such multiplications of the present, coming from the avant-gardes, that this chapter presents as a mode of fun.

To conclude, this book brings together a collection of reflections, histories and propositions from academic researchers alongside those of programmers, artists, curators, designers, art theorists and combinations thereof. We hope that in this movement across the fields in and around software the discourse collectively produced is varied enough to entice everyone to find something of fun for themselves in this book: let your curiosity cling, with avid fellow feeling, to its pages.

Notes

1 From the webpages of *LoveLetters* project by David Link (2009), http://www.alpha60.de/art/love_letters (accessed 12.01.2014); see also Strachey, Christopher, 'The 'Thinking' Machine', in *Encounter. Literature, Arts, Politics*, 13 (1954): 6, 25–31.

2 Link, David, 'There Must Be an Angel. On the Beginnings of the Arithmetics of Rays', in *Variantology 2. On Deep Time Relations of Arts, Sciences and Technologies*, (eds) Siegfried Zielinski and David Link (Cologne: König, 2006), 15–42, http://www.alpha60.de/research/there_must_be_an_angel (accessed 12.01.2014).

3 Cited in Link, David, 'There Must Be an Angel', 18, reference 8, as: 'Campbell-Kelly, Martin, Programming the Mark I. Early programming activity at the University of Manchester. *Annals of the History of Computing* 2 (1980): 130–68, p. 133'. Link further says: 'This excellent article describes the machine in detail. Strachey's accomplishment is all the more admirable since Turing's manual teems with mistakes and inaccuracies. It forced the reader to do some "de-bugging" immediately and painfully to complete a learning process of the foreign language. Cf. Frank Sumner, "Memories of the Manchester Mark 1"', *Computer Resurrection. The Bulletin of the Computer Conservation Society*, 10 (1994): 9–13, p. 9: 'So all the programmes written in Mark 1 code had slight errors in them, and by the time you had worked out what the code should have been you had become quite a competent programmer'.

4 Link, 'There Must Be an Angel'.

5 Gaponov, Yu.V. 'Traditions of Physical "Art" in the Russian Physics Community', Report at the Niels Bohr Archive Symposium, http://www.nba.nbi.dk/files/sem/symp/gap.html (accessed 12.01.2014).

6 It is interesting to consider this alongside a reading of the history of the scientific self as proposed in Daston, Lorraine and Galison, Peter, *Objectivity* (New York: Zone Books, 2010).

7 Fuller, Matthew, 'Elegance', in *Software Studies: A Lexicon*, ed. Matthew Fuller (Cambridge, MA: MIT Press, 2008), 87–92.

8 Knuth, Donald E., *The Art of Computer Programming, Vol. 1: Fundamental Algorithms* (Upper Saddle River, NJ: Addison-Wesley, 1997).

9 Coleman, E. Gabriella, *Coding Freedom. The Ethics and Aesthetics of Hacking* (Princeton, NJ: Princeton University Press, 2013).

10 Dijkstra, Edsger W., 'The Humble Programmer', ACM Turing Lecture 1972, EWD340, http://www.cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD340.html (accessed 12.01.2014).

11 Ensmenger, Nathan, *The Computer Boys Take Over* (Cambridge, MA: MIT Press, 2010).

12 Knuth, Donald E. *Selected Papers on Fun and Games* (Stanford, CA: Center for the Study of Language and Information, 2010), xvii.

13 From a conversation with David Link; also see Link, 'There Must Be an Angel'.

14 I am indebted to Celia Lury for this idea. Lury uses Rosalind Krauss's 'Video: The Aesthetics of Narcissism' (*October*, spring 1976).

15 David Harvey mentions the role of computational structures in the rise of neoliberalism in Harvey, David, *A Brief History of Neoliberalism* (Oxford: Oxford University Press, 2007).

16 Deleuze, Gilles, *Desert Islands and Other Texts 1953–1974* (Cambridge, MA: Semiotext(e), 2004), 270.

17 Caillois, Roger, 'Mimicry and Legendary Psychastenia' (English translation 1935), http://www.tc.umn.edu/~stou0046/caillois.pdf (accessed 12.01.2014).

18 Caillois, Roger, *Man, Play and Games* (Champaign, IL: University of Illinois Press, 2001).

19 *Software Studies: A Lexicon*, ed. Matthew Fuller (Cambridge, MA: MIT Press, 2008).

20 There is a wide range of literature that could be referenced here; see, for example, Uprichard, E., Burrows, R. and Byrne, D. (2008) 'SPSS as an "Inscription Device": From Causality to Description?', in *The Sociological Review,* 56 (4): 606–22.

21 See, for instance, Mackenzie, Donald, *An Engine, not a Camera: How Financial Models Shape Markets* (Cambridge, MA: MIT Press, 2008); Wilkins, Inigo, 'Destructive Destruction? An Ecological Study of High Frequency Trading', *Mute*, 22 January 2013, http://www.metamute.org/editorial/articles/destructive-destruction-ecological-study-high-frequency-trading (accessed 12.01.2014).

22 Turin, Luca, *The Secret of Scent. Adventures in Perfume and the Science of Smell* (London: Faber and Faber, 2006), 115.

23 Haraway, Donna, 'A Cyborg Manifesto: Science, Technology, and Socialist-Feminism in the Late Twentieth Century', in *Simians, Cyborgs and Women: The Reinvention of Nature* (New York: Routledge, 1991), https://wayback.archive.org/web/20120214194015/http://www.stanford.edu/dept/HPS/Haraway/CyborgManifesto.html (accessed 12.01.2014).

24 Edwards, Paul N., *The Closed World. Computers and the Politics of Discourse in Cold War America* (Cambridge, MA: MIT Press, 1997).

25 Castelle, Michael, 'Relational and Non-relational Models in the Entextualisation of Bureaucracy', *Computational Culture, A Journal of Software Studies*, November 2013, http://computationalculture.net/article/relational-and-non-relational-models-in-the-entextualization-of-bureaucracy (accessed 12.01.2014).

26 Deleuze, Gilles, *Nietzsche and Philosophy* (New York: Continuum, 2005), 4.

27 Berger, Peter L., *Redeeming Laughter: The Comic Dimension of Human Experience* (Berlin: Walter de Gruyter, 1997), 202.

28 Nietzsche, Friedrich Wilhelm, *The Gay Science* (New York: Dover Publications, 2006), 29.

29 Ibid., 31.

30 Ibid., 108.

31 See Kierkegaard, Søren, *Fear and Trembling, Repetition,* Kierkegaard's Writings, 6 (Princeton, NJ: Princeton University Press, 1983). See also Kierkegaard, Søren, *The Humour of Kierkegaard, An Anthology*, ed. Thomas C. Olden (Princeton, NJ: Princeton University Press, 2004).

32 Critchley, Simon, *On Humour* (New York and London: Routledge, 2002), 101.

33 Berardi, Franco 'Bifo', *Soul At Work* (Cambridge, MA: Semiotext(e), 2009).

34 Fuller, Matthew and Goffey, Andrew, *Evil Media* (Cambridge, MA: MIT Press, 2012).

35 Deleuze, Gilles and Guattari, Félix, *Anti-Oedipus: Capitalism and Schizophrenia*, trans. Robert Hurley, Mark Seem and Helen R. Lane (New York: Continuum, 2004).

1

Technology, Logistics and Logic: Rethinking the Problem of Fun in Software

Andrew Goffey

Introduction

Having emerged in the conflict-ridden, military-industrial complexes of the Cold War world, computation was not a likely contender as the technology most likely to mobilize affective energies in the frivolous, wasteful expenditures of pleasure. From the meticulously exacting requirements of its physical structures and logical architecture, through the disciplined hierarchies of its operation, to the thanatopolitics of its command-and-control end uses, there was perhaps little that was more refractory to the formal-material calculus of computation than *fun*. Whether one considers computation in terms of the development of something approaching a science (underlining its connections with changing paradigms of knowledge production), in terms of the less obviously but no less important history of bureaucracy,[1] or, indeed, from the point of view of the gender-freighted qualities of its division of labour, the initially predominant affective qualities of computer programming are *grey*, both ethically and aesthetically. Indeed, with its implication in the logistico-semiotic calculation of missile ballistics or workplace automation, as well as its eventuation in a software industry organized around business administration and data processing, computer programming seems to carry to an extreme an understanding of technology as a utilitarian tool predicated on a reframing of the world as a set of calculable quantities.

Framing accounts of the history of software in terms of the governmental 'mechanization of discourse' (Agar),[2] the technocratic logistics of a military-industrial complex or the intensification of Taylorist practices of 'scientific management', it is difficult to see how a story can be told about software that does anything other than ratify its heavily logistical and, at best, utilitarian underpinnings. As a set of technologies, techniques and practices centrally implicated in the reshaping of power relations in the workplace and beyond, this becomes something of a problem, particularly for accounts of computation that might seek to address ways to escape from the unreasonable exactness of algorithms and the implication of computational technologies in the extension of logics of state and work to society more broadly. And yet, as a number of accounts of the development of the computer games industry have pointed out,[3] from an early stage in its modern development, computing has not only known a series of more or less surreptitious subversions of these dominant trends in its development but has also, in debates around what programming *is* or *might be,* demonstrated an interesting refractoriness to purely formal-rational understanding.[4] Considered as an *art* or at the very least as *artful,* there are aspects of computer programming that require different historical reference points and conceptual framings in which the literal and metaphorical margins of the development of programming and programming practices start to point towards a way of thinking about their technologies.

Drawing on an alternative set of conceptual resources, this chapter offers a theoretical discussion of how one might start to address the problematic question of fun in software. It asks whether *in principle* it is possible for software to become a technology of enjoyment. In practice, of course, software and the computational technologies it powers has become a central element in the reshaping both of leisure time pursuits and of work as something that should also be 'fun'– not least of all in the tech sector itself.[5] So, the question might sound a little odd, especially given both the rigorously formal-logical nature of computation as it appears in dominant accounts and the historically predominant sense of software as a technical embodiment of an epistemological worldview. Yet there is a philosophical – even speculative – element to computation, both in theory and practice; and the recurrent references to ideas of art, to virtuosity, even to free speech that can be found in accounts of software and its development call for an understanding that entails addressing it in terms that are more nuanced than the formal-logical categories of mechanized mathematics.

In software

Given the specifics of the history of the development of computing and software, it is hardly surprising that formal reflections on the place of fun, pleasure and emotion in computing came along somewhat late in the day. In the field of research into human–computer interaction, 'fun', or variants thereof, appears not to have started to become much of a consideration until the early 1980s. By the time Carroll and Thomas published their well-known article on fun,[6] the modern computer had already acquired a familiar place within the workplace and was starting to make serious inroads into the domestic spaces of Western nations. The pleasures of computer games – at that point largely confined to arcades – had been recognized and some of their potentials cannily noted among researchers eager to understand and replicate the interactional successes of the gaming interface.[7] When the hierarchical structures of work and the pragmatic necessities of discipline could not be relied upon to compel obeisance, and increased processor speeds made it possible, it was perhaps not surprising that experts in the field of human–computer interaction might seek to explore aesthetic questions of affectiveness and pleasure as a bolt-on motivator to make personal computing software packages more appealing. Fun, in this optic, was not simply confined to gaming but was crucial to subjective judgements of usability. Read outside of the psychologistic framework in which the early 'funologists' understood it, the identification of fun as something that would potentially make equivalent tasks more interesting[8] sounds mildly cynical. While they later acknowledged that using the fantasy element of games to make information processing jobs more interesting, 'to overcome the boredom and vigilance problems' that inhere in routine work, might seem 'exploitive [sic]' to workers, Carroll and Thomas's early proposals for building 'fun' into computing initiated something of a trend to blur the boundary between 'tool' and 'toy' that continues apace.[9] In any event, it is difficult not to read some of these studies as at best market research or advocacy of methods for keeping a workforce happy, and therefore not really as rigorous explorations of the pleasure and pain of code.

However, rather than seek to address the broader implications of the way that 'funologists' sought to inscribe enjoyment within the interfaces of computer programs, this chapter looks to contest the implication of what we have contended so far, i.e. that fun is somehow anathema to the essence of computation. Close consideration of computational devices and a cursory understanding of the onion-like layering of abstractions that distance the programmer and, even more so, the end-user, from the underlying architecture of a machine that requires the minutest and most exacting of instructions about what to do, *do* tend to suggest that pleasure could only ever be a perfunctory addition, a token consideration, for a subject implicated as the last stage in a set of machinic operations from which he/she is *architecturally* excluded and increasingly distant. If there is subject of enjoyment in computing, he/she might, from this technical point of view, be considered the mere residue of a technological infrastructure to which she can only relate by accepting her position as its potentially untrustworthy – and typically presumed stupid – end point. Between the output of some computational process received by him/her as stimulus and the input that he/she provides to some other such process as a response, the end-user would in this view merely form a behavioural arc feeding the machine with data to process, as he/she processes the data it supplies in return.

To avoid the problem of making 'fun' an issue for more sophisticated user acceptance testing, the question needs to concern the kind of affordance, within the technical logic of software itself, that might allow us to make more sense of the idea of fun *in* software, more sense of the possibility of their being some kind of intrinsic connection between the formal programmatic structures of code and the possibilities of enjoyment, or if not an 'intrinsic' connection, at least a connection that does not rely on an individualized psychology in which the software itself is largely irrelevant. What, indeed, might be the computational determination of fun? One may take it as read that there is often an intense investment of programmers in their work, and a pleasurable aesthetic to their practices.[10] Yet it is all too easy to focus on the end points of that enjoyment, the human element that is presented with the finalized artefact, and overlook the broader set of arrangements, the assemblages, of which that element is a part, and thus fail to consider the broader socio-technical determinations of the politics of enjoyment at work here. Focusing on individual psychology misses the affordances of code itself as well as the broader organizational context within which software is operative. Considering computers as media, for example, makes it a little too easy to pour old wine into new bottles and to understand the subjects of computation as the end points of a chain of communication, a consumer or at best an interactive audience member, but this would miss crucial aspects of the ways in which fun or enjoyment lurks quite crucially within computation, pointing towards an aspect of the software that is otherwise readily overlooked.

A technology of enjoyment?

As has already been pointed out, the rather desultory, belated development of 'funology' as an approach to exploring frivolous delight in the workings of software is no surprise, given the genealogy, the lines of descent, of software.[11] Equally, it is of little surprise that figuring out the place of enjoyment in the interactions between humans and computers should pose such thorny conceptual problems when they *are* tackled. Given the historical connections of HCI with the experimental psychology and behaviourist paradigms of the kind of cognitive research that informed technology developments in the US military after World War II, the research labs of the burgeoning giants of corporate computation,[12] as well as the framing of its agendas in terms of a widespread efficiency concern over how to make computers more useable, it is difficult to see how one could avoid thinking through the pleasures of computation in terms of an underlying set of assumptions about efficiency and utility. But one should not be too quick in rushing to blame the psycho-logistic framework of HCI research here for making such assumptions. Broader reflections in the philosophy of technology more generally make it clear to what extent thinking about technology is imbued with an instrumental-rational, utilitarian calculus concerning useful work. Indeed, as the German philosopher Gernot Böhme has argued, the self-evident connection between technology and usefulness has become so engrained that we find it difficult to understand technology in 'any other way than in terms of the particular goal it serves',[13] a conceptual problem that Böhme attributes to Marx. Yet Böhme claims that the 'intrinsic relationship of technology and usefulness is a product of the theory of technology, not of its actual history',[14] a view that should interest us here.

Developing a line of thinking that directly implicates computation in economic strategies of exploitation and control, George Caffentzis has proposed a reading of the logic of the Turing Machine that suggests that computation is intrinsically bound up with the capitalist understanding of technology as about work.[15] In Turing's account, qua logical model, a computer program is a materially effective representation of some real-world entity or set of entities. Qua real-world entity, a computer program is a materially effective replication of human labour, each execution of which is formally equivalent. And the possibility that the computer program presents of the perfectly calibrated and calculable repetition of a pattern of action, without tempestuous militancy, absenteeism or complaint, makes it ideally suited, as dead labour, to replace human work. To the extent that computerized technical machinery has historically been centrally implicated in substituting for human labour, prolonging a major tendency of the capitalist mode of production, with its replacement of living by the dead labour of machines, Caffentzis's account seems fundamentally correct, and it indirectly confirms Böhme's argument, at least in part. However, while such an account, at a certain level of abstraction, makes good sense, one might question whether or not it tells us everything, whether the means-ends rationality it presupposes is adequate for an exhaustive understanding of the technologies of computation. The formal structure of the Turing Machine tells us very little about what programming as a practice is, how computational structures function in practice or indeed how work is transformed by machines, except at the most general of levels.

Böhme's broader historical contextualization of understandings of technology questions the way in which it has predominantly been framed in the utilitarian terms of its position in the instrumental practices of means-ends rationality. Indeed, the ancient Greek term 'mekhane', from which the modern term 'machine' derives, displays a range of meanings of which the idea of a 'useful effect' is only one. Discussing Salomon de Caus's *Of Forceful Motions. Description of Divers Machines Both Useful and Enjoyable* in precisely these terms, Böhme has argued that alongside classic examples of technology adduced in favour of the instrumentalist view – mining, metallurgy, construction and war – there are myriad examples that demonstrate the existence of playful technologies, aimed, as he puts it, 'at instruction and entertainment'.[16] Technology is not always and everywhere about extracting useful movements from nature. Historically it has sometimes been about movements that are 'para phusin'– i.e. that run counter to nature – and this suggests that a fuller understanding of such technology and its history requires situating in the context of a different kind of economic thinking.

Where *useful* technologies, in Böhme's account, developed in tandem with the bourgeois ascendancy of the nineteenth century in the emergence of modern industrial capitalism, *enjoyable* technologies developed principally in the courts and palaces of Europe. For Böhme, these are two distinct historical lineages that have become confused in contemporary understandings of technology. Wasteful expenditure on the ridiculous or absurd is the key historical indicator that one is dealing with a technology of enjoyment. As Böhme puts it,

> a great many technologies were originally developed for purposes of display, popular entertainment, prestige or sport before any potential for a useful, i.e. economically profitable, application could be found in them. Air travel, electronics and the automobile all spring to mind.[17]

While one might not want to contest the historical record that demonstrates the obvious connection between computation and work, as in, for example, its pressing into service for workplace automation,[18] Böhme's insistence that there is another line of descent in technology than the utilitarian might offer an interesting lens through which to reconsider computing and the technological aspects of the military-industrial complex that developed in post-war America. Indeed, Georges Bataille – whose ideas Böhme draws on – sees American post-war economic development (its massive investment in military material) as exemplifying the problem of 'general' economy, that of the expenditure or discharge of energy (not its extraction), arguing that the US economy is an economy that 'is in fact the greatest explosive mass the world has ever known'.[19] The fascination with automata evinced within Department of Defense-funded research in artificial intelligence (AI) could perhaps be adduced as evidence that technologies of enjoyment were alive and well at the heart of the strategic considerations of the US military machine, forming something of a compromise solution to the useful/wasteful distinction noted by Böhme, represented as part of the pursuit of a techno-scientific ideal, functioning as a sponge for excess energies.

It is thus important that one at least acknowledges that the 'useful' is not the only way in which to frame an understanding of technology, any more than is 'imitating' nature. Of course, as the example of AI suggests, in practice, it is exceedingly difficult to make clear-cut distinctions between the useful and the enjoyable. Indeed, the sumptuary expenditure of the post-war military-industrial complex, out of which so much early development in software emerges, seems to point instead towards a kind of monstrous enjoyment in the excessively and lethally functional, or better still, suggests a search for ways in which to functionalize the flows of excess energy generated as part of this general economy of which the new technologies of computation were to become such a significant part. So, what is required here are some more refined conceptual resources for helping us think through the ways in which computational technologies and the practices associated with them do not fulfil their heavily codified, highly functional remit.

Ineffective thought processes

An understanding of technology in terms of the useful is a deep-rooted and enduring legacy of the growth of capitalism and it is one that frames, implicitly or explicitly, much work both on software and on the historical development of computer programming. However, it is important to bear in mind the possibility of an alternative reading, and it is towards a development of this alternative that I would now like to turn, using Marvin Minsky's 1980 paper on 'Jokes and Their Relation to the Cognitive Unconscious'[20] as my starting point.

The interest of Minsky's essay in this context is considerable, because his is an account that is heavily, if not explicitly, informed by the material state of computational technology and the practices associated with it. By addressing the question of humour in direct relation both to Freud and to computation, he offers us a way in to considering more directly the question of how we might understand the connection between enjoyment and software, and this time, not quite from the point of view of an end-user whose discontented, workplace-induced boredom must be countered.

Written at approximately the same time as the first studies discussing the place of fun in software, Minsky's essay seeks to complement Freud's account of humour in terms of a breakdown of the censoring of affectively charged contents of unconscious thought processes by means of a more direct consideration of reasoning as such and the nature of intelligence. Minsky argues for the existence of a *cognitive* unconscious and for an understanding of humour that provides it with an important function in relation to the faulty reasoning of commonsense. In his account, humour and the pleasures it affords are to be understood in evolutionary terms, as an outgrowth of the keying in of the neural impulses of cognition into survival and the circuits of stimulus and response characteristic of *behaviour*. Jokes and, more specifically, logical paradoxes in this respect have survival value. Drawing on his own well-known theory of frames to explore the problematic nature of commonsense reasoning, Minsky argues that, viewed in terms of cognition, the role of jokes and humour is to help us avoid the potentially destructive consequences of the errors – or bugs – to which such reasoning is prone. From his point of view faulty reasoning is just as 'naughty' as the repressed wishful thinking that is central to Freud's view of the unconscious, and calls out for a similar kind of censorship, albeit for reasons of the survival of the species.

Setting aside his rather typical and heavily utilitarian understanding of evolution (in which life forms evolve exclusively in terms of adaptive self-interest),[21] what is interesting and important about Minsky's argument, at least for the purposes of this chapter, is the way in which it allows us to make a connection between fun, computer programming practices and the structures of software. Although his essay begins with a reference to the well-known *logical* paradox of the Cretan Liar, it is largely in terms of the 'bugs' evident in commonsense reasoning that Minsky seeks to develop his argument. Bugs, as he puts it, are 'ineffective or destructive thought processes' and they arise when the cognitive censors that prohibit us from following cognitively 'harmful' paths of reasoning fail. A logical paradox such as that of the Cretan liar, who states 'I am lying', of course leaves us puzzling, and is not in and of itself destructive, but for Minsky it points towards the thin ice of 'self-reference' and the unproductive meandering of faulty reasoning. That such reasoning *might*, in his words, do us 'some vaguely understood cognitive harm' makes sense, but the leap from there to 'ineffective or *destructive* thought processes' (emphasis added) could be considered overkill, or as pointing towards the possibility that his text is addressing a broader problem than might at first seem to be the case. Indeed, it is not even evident that the paradox of the liar posed quite the problem in its original context it has done for modern logicians.[22] However, Minsky's point is that like the bugs to which it is related, the self-referentiality of the liar paradox has the propensity to leave us spinning in circles. And yet there are contexts, of course – that of a computer program used to guide missiles, for example – in which this kind of self-referentiality could indeed be destructive, and there are people – computer programmers – for whom bugs are indeed a bit more than a simple intellectual problem. In this respect, the reference to 'bugs' as 'ineffective' or 'destructive' in Minsky's account suggests that there is a much more concrete, technological aspect to the problem with which he is concerned. Characterizing commonsense reasoning as 'buggy' makes perfectly good sense if the underlying frame of reference of one's argument is computer technology (and not simply AI). Indeed, while one might take the broad sweep of the claims made about a cognitive unconscious at face value and accept the validity of Minsky's qualified characterization of reasoning in terms of the efficacy of AI 'goal structures' and 'plans', his further references to 'administrative structures' do rather compound the impression that for Minsky the problems of the cognitive unconscious are to be understood against the background of the kind of technocratic, command-and-control horizon that accompanied AI from the outset.[23]

Framing the bugginess of commonsense reasoning and the humour it generates in terms of the effective and the ineffective tends to suggest, then, that being 'inscribable within the formal-material calculus of computation', or more bluntly 'programmable', is the ultimate arbiter of what is and is not reasonable. But the broader issue is that, with his goal structures and plans, it is evident that for Minsky the enemy is thinking that is without purpose, 'unproductive' thinking, an enemy that must be actively and vigilantly combated. Play and the pleasures of humour are of value only in so far as they facilitate the recognition and avoidance of bugs, and the point of code – whether glossed as the effective commonsense reasoning of individual psychology or the axiomatic structures of AI – is to serve some sort of higher end. As he puts it pithily, 'jokes are not really funny at all, but reflect the most serious of concerns; the pursuit of sobriety through the suppression of the absurd'.[24] Left unchecked, commonsense reasoning might dissolve in absurd and pointless puzzles, so it is better to recode humour within the functional calculus of the work of programming.

Minsky's essay is of interest then because if it is read as dealing with the technological issue of how to write effective computer code, it can be construed as developing a concern about the appropriate disciplining of thought among computer programmers, as much as being about cognitive processes. Like the computational psychology that developed concurrently with the rise of computation, there appears to be something of a technological underpinning to the pleasures and pains of the unconscious in Minsky's text, an underpinning that points to the broader social and political context of production. Without greater subjection to the command and control structures exemplified in the closed world of AI, computer programmers would not only spend even more time debugging code, they might become ineffective and create problems in mission-critical military software.

While such a reading does a little violence to Minsky's text, it makes good sense from the point of view of the sociology of knowledge, because it suggests his concern to make a tacit case for the continuing investment in academic computing and more specifically in AI research. Yet the account also points up a problem that has been recognized in more historically contextualized accounts of the development of programming as an activity. What kind of intelligence is needed in order to be a programmer, what kind of thinking is best adapted to the structures of software, and must programming ultimately be informed by the diktats of technoscience? Questions about what made a good programmer were common with the emergence of the software industry, and the theme of programming as a 'black art' was not uncommon in the 1960s, before computing science as a properly academic discipline had emerged. Ensmenger has underlined the extent of discursive connections between the programmer and the poet, both of whom are seen to share a demiurgic (or, better, semiurgic, as working with signs) capacity for conjuring up something out of nothing. And in this respect, Fred Brooks's reference to the 'exertions of the imagination' of the programmer (in his classic text *The Mythical Man-Month*) reminds us of the tenacious insistence of the aesthetic on the fringes of the rational under conditions of the latter's instrumentalization.[25] Taking these references together with Minsky's later opening up of the question of humour, one might want to look for a rather different reading of the pleasures and pains of programming, a reading that, while more openly speculative, takes the question of the fun in software in different directions.

Coding for coding's sake

In the world of computer programming, the computational equivalent of paradoxes like that of the Cretan liar would perhaps be something like an infinite recursive loop. In this respect, the uncontrolled use of self-referential statements is obviously problematic, something that undergraduate students learn in Programming 101. But technological determinations of the problem of self-reference aside, there is nothing particularly new in the kind of anxiety that Minsky feels – and thinks that others should feel too – over the ambiguities and inconsistencies in logical, or perhaps, semi-logical thought. In the wake of Frege and Russell, developments in symbolic logic in the first part of the twentieth century exemplified a suspicion, even hostility, towards the natural languages in which such paradoxes as that of the Cretan liar are uttered. Alfred Tarski, for example, was moved to claim that there is something 'evil' about such languages, thereby pointing to the difficulties to which their *free* use can give rise.[26] The problem with natural languages is that they offer little in the way of well-regulated control, and more particularly, do not admit the kind of stratification of levels of languages that, by differentiating between object language and meta-language, would disambiguate self-reference. One cannot avoid the problems that speaking for the pleasure of speaking generates other than by a translation of one language into a meta-language that re-reads one's statements as making some sort of logically disambiguated truth claim. Yet with Minsky, who does not frame the issue as a purely logical one, we learn that the situation is a little more complicated than it might at first appear. It is by means of the kind of paradoxes that natural language excels in generating (and which formal systems, albeit in a very different way, as we know since Gödel, cannot dispel) that one can learn to avoid the useless, purposeless thinking that becomes so ruinous in the development of computer software, putting algorithms and data structures at risk of wasting memory, never halting, or grinding to a halt where and when it is least expected. But we could equally read such paradoxes as pointing towards a kind of 'surplus value' of code, the idea that the statements of a language (whether natural or formal) bear within them a margin of possibility for use that can not only upset the existing norms which govern the way that language is or can be used, but generate alternative kinds of functioning. In this sense, what logicians are inclined to see in terms of ruinous deductive inconsistency might be treated as pointing towards a source of change and invention that formalization *per se* struggles to understand.

In this respect, the broad intuition, that logical paradoxes, jokes and nonsense tell us something important about computational intelligence is correct, but only partially so. There is an unthought connection, which Minsky's text points towards, between the kinds of logical paradoxes explored in Ancient Greek philosophy, Freud's understanding of humour and the possibility of coding errors. Humour and the pleasures it generates, it might be suggested, arise where codes are played with for coding's sake, where the references of statements recede in importance and the ordered norms of cognition lose sway. In this respect, the possibilities of the surplus value of code glimpsed in paradox and combated in bugs, suggests a way of understanding programming practices and the development of computational forms that are not slavishly submitted to 'goal structures', 'plans' and 'administrative structures'. The development of programmatic structures without real-world referents, the possibilities of simulations without any models, might from this point of view, be considered key aspects of a 'black art', an 'exertion of the imagination', which a strictly cognitive point of view, anxious about any excess of code not correlated in a model referent, does not clarify. It is tempting, in fact, to point towards the need for a strictly aesthetic reading of this aspect of computation and of the place of fun in software it illuminates. If the project of AI was, as Joëlle Proust has claimed, ultimately, a philosophical one, analogous to that of Kantian philosophy,[27] Minsky's bugs, and the problematic situation of pleasure in relationship to computation they point to, lead us straight from software to earlier historical marginalization of aesthetics and enjoyment in philosophy. From this point of view, the difficulties of conceptualizing the place of fun in software become a little clearer – the recurrence of some kind of aesthetics of experience is almost inevitable when one set of fundamental standards – such as that which is entailed in computational formalism – seeks to impose itself across the board. Aesthetics, and the painful and pleasurable qualities of the affective experience from which it arises, is necessitated where cognition fails to capture the 'wild' things of the world in determinate concepts,[28] and in this domain there are 'no rules, methods, foundations, or criteria'.[29]

But what happens when code is generated for coding's sake? And what might that code consist of? Here the heavily science-inflected accounts of computation that stress the way in which it models real-world processes hamstring us. Coding for coding's sake entails exploiting the 'play' inherent in the structures, types and other elements of computer code that enables them to be turned to other, perhaps less work-like, ends. It suggests that the dominant view of computation only captures part of the story about software, the part that connects most easily with the generally utilitarian understanding of technology examined earlier in this chapter, an understanding on which it has conferred considerable theoretical sophistication, as exemplified in notions of techno-science, cognitive capitalism, immaterial labour and so on. Reframing computation and the question of fun in software in terms of code or coding for coding's sake shifts our attention. The particular abstracted theoretical terms of the dominant view tend to preclude any appreciation of the material specificities of practice or the technical details of structure and function (mere implementation details of mechanical-mathematical formulae). By contrast, considering the question of fun in software in terms of coding for coding's sake brings the affective and the material back into the picture, linking technical structures with everyday practices. The de facto, historical positioning of computing and software engineering as instrumental practices does not in and of itself ground what these practices might be, or might become, and reading coding as ultimately cognitive obscures forms of practice that can then only be articulated as a 'black art'. Exploring software in terms of the 'play' of code, the excess of possibility it bears within it, is a way of taking this idea of a 'black art' seriously.

Enjoying artefactuality, or the politics and pleasures of free speech

By reframing the problem posed by Minsky as one that arises because of the existence of a kind of 'play', a margin of 'decoding' inherent in any kind of code – manifesting itself in natural language in the paradoxes of self-reference, in formal logic as incompleteness – we have a way back to the problem of fun in software. As software development became more sophisticated in the 1970s and into the 1980s, it becomes less plausible to maintain the view that what computer programs do is provide a model of some real-world process or set of processes. To put this in slightly different terms, while much early work in software engineering was indeed about modelling real-world processes (whether that be developing missile guidance systems, airline reservations or some other logistical command-and-control system), the functions and routines, algorithms and data structures that make up programs and programming possess a margin of decoding that places them in inherent excess over the well-structured and organized, useful models of real-world functionality. And this, in turn, makes possible those aspects of computation that cannot be understood simply as about the extraction of some useful effect from the existing world, returning us to that understanding of 'mekhane' as a movement against or beyond nature, 'para phusin', that was mentioned earlier in relation to technologies of enjoyment. Computer programs are for the most part heavily functional and decidedly functionalized entities, both internally and externally, in the sense that they are largely designed for specific purposive activity, to do specific things, and in order do these things must be carefully organized as a set of algorithms and data structures that function effectively. However, there are differences in the kind of functioning that might be obtained from and through computer programs, qualitative differences that the strangely abstracted idea of software as modelling real-world processes allows us to ignore.

Strictly speaking, many of the things that software models are not real-world processes at all, they are things that are brought into being by computational technologies themselves. Pointing with a mouse or using a joystick might be a bit like real-world processes and they might be implicated in practices that seek to replace real-world processes, but we miss what is specific to what software itself does if we make some set of assumptions about its successful connection to real-world processes our starting point. One might consider that writing on a screen using a stylus captures enough of the 'essentials' of what writing is to make a claim about modelling plausible, but one would have to forget a great many concrete aspects of the experience of writing, as well as the other technologies to which it is connected, to do so. Similarly, as hacker history shows us, many of the early developments in programming as a set of practices arises from an exploration of the technologies themselves, for their own sakes. More pointedly, by focusing on the explicitly formalized and the cognitive, it becomes easy to forget that a great deal of the work of programming is ultimately a work on and with sensibility, a kind of activity that is tacit and cannot be explicitly formalized. In this respect, the question of fun in software, and the limitations in cognitive and/or formal-logical accounts of computation that have been explored here, suggests the need for some fine-grained distinctions in the way in which one attempts to account for the connections between technology and enjoyment. There is an element of artifice within software that becomes invisible when it is approached from within the epistemic framework of logic, at the broadly 'molar' level of organized, functional purposiveness. The 'demiurgic' quality of software noted by Brooks, its capacity to produce something that did not exist before, entails a more detailed focusing on 'molecular' processes that are necessarily brought into play in the generation of code: clever tricks with assembly language, manipulating the side effects of an algorithm, devising workarounds, the smoke and mirrors of user interface design, and so on. The virtuoso smarts of programming practice that get software working can of course eventually be explained logically if the code compiles, the app works and users accept it. But in doing so, the process disappears into the product and the experimenting, the 'bricolage' gets forgotten.[30] And similarly the algorithmic operations of the code itself, when software is used, are not exhaustively explained by the tasks for which they are devised and employed. Anyone can get hypnotically captured by the pleasures of algorithmic repetition, endless scrolling or clicking-through, linking, ticking, pointing, clicking … The point is that the technical affordances of code, its margins of possibility, the practices that follow these possibilities in the construction of software and the broader connection between software and the real-world situations in which it eventually gets used entail a kind of functioning that operates aesthetically, a functioning that is difficult to understand within the dominant ways of exploring computation.

However, those aspects of software that are marginal in the formal-logical, epistemological views of computing science take on a major importance when considered in other fields of knowledge and other kinds of practice. It is not just that the virtuoso pleasures of programming practice become more readily visible when one casts, say, an ethnographic eye over open-source software development – as E. Gabriella Coleman has done – but that structurally important aspects of software as such come to the fore. Legal debates around software considered in terms of freedom of speech suggest that a politics of rights, albeit a liberal one framed in terms of US law,[31] recognizes precisely what formal logic prefers not to: that computer programming must be understood as a kind of *enunciative* process. Free software, such as the Debian distribution of the Linux operating system discussed by Coleman, is legally analogous to the freedom of speech protected by law in the United States by the First Amendment. The development of licensing practices that allow source code to be used, modified and distributed 'as if' that code were 'actual' speech discloses a broader fact about the formal-material code of software: it is a set of 'utterances' that take place in the world, that are connected to agents who 'do' that uttering. The fact that the history of software and its development is one that has been dominated by proprietary jurisdiction that ties software in to heavily commercial, highly capitalised organizations should not preclude recognition that software is a semiotic artefact, a set of operations in and on codes that implies the ongoing, repeated fact of enunciation. Something gets 'expressed' in software, and legal discourse around the First Amendment captures and frames it, as Coleman puts it, in terms of 'personal control and autonomous production'.[32] US law then, is attentive, in its own individualistic way, to a dimension of software – its quality as enunciation – to which coding for coding's sake has also drawn our attention, because the kind of play that we have argued becomes visible with paradox is closely linked to enunciation. Contemporary logic has a tendency to exclude the 'fact' of speaking, uttering or articulating from its framework, or at least to capture it in its own terms. To acknowledge enunciation *per se* risks bringing back into the codes of logic something – often a 'subject' that would threaten the absoluteness of the logic one is advocating. The claim of formal logic (and hence the explanatory ground of computation) is to have escaped the experiential starting point presupposed in natural language. Yet as has been seen, enunciation cannot really be so easily excluded. The problem of self-reference that was discussed earlier demonstrates this readily. Enunciation is problematic for theories of language and logic that seek to frame those languages – formal or otherwise – in purely self-consistent ways, because the fact of enunciation, of the act of speaking, uttering, inscribing, marks what is spoken, uttered, inscribed with an excess that cannot be captured within formally explicit structures. In this respect, relying on a formal-logical account of software precludes the development of an attentiveness to both the enjoyment that is afforded within and by software, the 'expressive' dimension of the experimental exploration of the possibilities inhering in coding languages, structures and technologies, and the politics of it. Yet law thinks otherwise, and it feeds back into programming practices, adding an interesting constraint on the way in which free and open-source software gets developed. Enunciation brings the messiness of the world back and forces us to connect coding or programming with practice in more rigorous ways. Indeed, it is precisely to the enunciative dimensions of programming that the virtuosity that is so frequently extolled within programming communities draws our attention. Programmers get a kick out of the code they write and the aesthetic qualities it displays. But at the same time, that aesthetics, and the pleasure and pain that comes with it, forms the starting point for an investigation of the politics of software freed from the epistemological trappings of modern techno-science.

Conclusion

This chapter started by pinpointing some of the historical and conceptual obstacles to developing an understanding of the place of fun in software. Those obstacles, it was suggested, necessitate a somewhat speculative, philosophically oriented set of tools to make it possible to develop an understanding of the connections between computational technologies and enjoyment that does not marginalize the technical as such. Unfortunately for the question of fun in software, it is rather too easy to rationalize away those aspects of computation that can't be readily parsed in terms of the discourses of computing science, because computing science typically provides a broad set of abstractions for the selective exploration of programming and for software engineering more generally. One of the consequences of this has been that the question of fun and enjoyment in computing more generally has been marginalized and misunderstood in terms of individual psychology: it is not easy to think through the set of processes – both technical and social – that shape the possibilities of enjoyment in computational technology, when one's conceptual framework abstracts pleasure and pain into subjective judgements of usability. The philosopher of science Isabelle Stengers has argued, with regard to modern science more generally, that 'the favourite vice of our tradition [is] to construct a perfectly convincing argument that, as if by chance, has the capacity to dissimulate or condemn a question that it doesn't feel very certain about'.[33] This is a claim with direct implications for the way in which software and software development gets understood, as one layer of formal-logical abstraction after another shapes the increasingly sophisticated Taylorist scrutiny that team leaders, project managers and consultants bring to bear on the development of software. It would be easy at this point simply to conclude that as far as fun in software goes, the possibility of a different reading of programming practices and coding structures – one in which enjoyment is more than a little bit of an add-on, a sweetener to make exploitation easier to bear – has missed the point. Indeed from the point of view of the hyperfunctional quality of many digital technologies, this has a plausibility that borders on the self-evident. Yet something does not quite add up in the techno-scientific worldview, and the abstracted starting point it offers for considering computation makes it easy to forget the ways in which code and coding practices form matters of expression in their own right, which can be played with, twisted and bent beyond the diktat of instrumental norms and so lend themselves – in principle, if not always in practice – to frivolous pleasures that might escape the unreasonable exactness of algorithms.

Notes

1 The writings of Agar and Cortada are useful in this regard. See Agar, Jon, *The Government Machine. A Revolutionary History of the Computer* (Cambridge, MA: MIT Press, 2003) and Cortada, James, *Before the Computer: IBM, NCR, Burroughs, and Remington Rand and the Industry They Created, 1865–1956* (Princeton, NJ: Princeton University Press, 1993); Cortada, James, *The Digital Hand: How Computers Changed the Work of American Manufacturing, Transportation, and Retail Industries* (Oxford: Oxford University Press, 2003).

2 Agar, *The Government Machine*.

3 See, for example, Dyer-Witherford, Nick and de Peuter, Greg, *Games of Empire. Global Capitalism and Video Games* (Minneapolis, MN: University of Minnesota Press, 2009).

4 With regard to the latter point, see the interesting account of early programming proposed in Chapter 2 of Ensmenger, Nathan, *The Computer Boys Take Over. Computer Programmers and the Politics of Technical Expertise* (Cambridge, MA: MIT Press, 2010).

5 See, for example, Ross, Andrew, *No Collar: The Humane Workplace and Its Hidden Costs* (Philadelphia, PA: Temple University Press, 2003).

6 See Carroll, John M. and Thomas, John C., 'Fun', *SIGCHI Bulletin*, 19(3) (1988): 21–4.
7 Malone's early paper and the PhD it was based on are fairly explicit in the way that they frame the interest in fun and pleasure in relation to the marketization of software.
8 Carroll and Thomas, 'Fun'.
9 For a relatively recent discussion, see Monk, Andrew, Hassenzahl, Marc et al. (2002), 'Funology: Designing Enjoyment', in *CHI '02 Extended Abstracts on Human Factors in Computing Systems* (New York, NY:ACM), 924–5. Doi=10.1145/506443.506661 http://doi.acm.org/10.1145/506443.506661 (accessed 08.01.2014).
10 Gabriella Coleman has offered an interesting account of some aspects of this issue in her book on open source: Coleman, E. Gabriella, *Coding Freedom. The Ethics and Aesthetics of Hacking* (Princeton, NJ: Princeton University Press, 2013).
11 Blythe, Mark A., Overbeeke, Kees et al., *Funology. From Usability to Enjoyment* (Dordrecht: Kluwer, 2004).
12 In addition to the software industry focused work of, for example, Martin Campbell-Kelly, Margaret Boden's study of the history of cognitive science, *Mind as Machine* is a useful reference in this regard. Campbell-Kelly, Martin, *From Airline Reservations to Sonic the Hedgehog. A History of the Software Industry* (Cambridge, MA: MIT Press, 2004); Boden, Margaret, *Mind as Machine. A History of Cognitive Science*, vols 1 and 2 (Oxford: Oxford University Press, 2006).
13 See Böhme, Gernot, 'Technical Gadgetry: Technological Development in the Aesthetic Economy', *Thesis Eleven*, 86 (2006): 54.
14 Ibid., 54.
15 Caffentzis, George, *In Letters of Blood and Fire. Work, Machines and the Crisis of Capitalism* (Oakland, CA: PM Press, 2013), 139–200.
16 See Böhme, *Mind as Machine*, 56.
17 Ibid., 62.
18 See Noble, David, *Forces of Production* (New Brunswick, NJ: Transaction, 2011).
19 Bataille, Georges, *The Accursed ShareI,* vol. 1, trans. Robert Hurley (New York: Zone, 1988), 171.
20 Minsky, Marvin, 'Jokes and Their Relation to the Cognitive Unconscious' (1981) in *Cognitive Constraints on Communication: Representation and Processes*, (eds) Lucia Vaina and Jakko Hintikka (Dordrecht: Reidel, 1984). See also Minsky, Marvin, *The Society of Mind* (New York, NY: Simon & Schuster, 1998).
21 The 'utilitarian' qualities of the Darwinian strands of modern biology have been criticized at length. See, for interesting commentaries on evolution and adaptation, Sonigo, Pierre and Stengers, Isabelle, *Cosmopolitiques* Vol. 1 (Paris: La Découverte, 2003) and Ansell-Pearson, Keith, *Viroid Life: Perspectives on Nietzsche and the Transhuman Condition* (London: Routledge, 1997).

22 See the discussion in Imbert, Claude, *Phénoménologies et langues formulaires* (Paris: Presses Universitaires de France, 1992).

23 Given the time at which Minsky was writing, this should perhaps not be a surprise – the growth of the software industry, and in particular of personal computing, suggests a certain defensiveness with regard to ordinary use outside of the institutions that had generally benefited heavily from government funding.

24 Minsky, Marvin, 'Jokes and Their Relation to the Cognitive Unconscious', see also http://web.media.mit.edu/~minsky/papers/jokes.cognitive.txt (accessed 08.01.2014, unpaginated).

25 Brooks, Fred, *The Mythic Man-Month. Essays on Software Engineering* (New York: Addison Wesley, 1975). In this respect, it is not at all far-fetched to recall the emergence of the Romantic lionization of the imagination against the excesses of scientific reason in the nineteenth century. See Williams, Raymond, *Culture and Society* (London: The Hogarth Press, 1987) for a broadly historical account.

26 See Tarski, Alfred, 'Truth and Proof', in *A Philosophical Companion to First-Order Logic*, ed. R. I. G. Hughes (Cambridge, MA: Hackett, 1993).

27 But reframed as a search for the 'transcendental' conditions of *all* cognition, see Proust, Joëlle, 'L'intelligence artificielle comme philosophie', *Le Débat*, 47 (1987): 88–102.

28 A lengthy parenthesis ought to be opened up here regarding the relationship between the aesthetic and the cognitive. But this would entail engaging in detailed discussions about the importance of Kant in relation to logic as well as philosophy more generally. For some interesting relatively recent discussion of Kant's aesthetics, see Gasché, Rodolphe, *The Idea of Form. Rethinking Kant's Aesthetics* (Stanford, CA: Stanford University Press, 2003) and Shaviro, Steven, *Without Criteria, Kant, Whitehead, Deleuze, and Aesthetics* (Cambridge, MA: MIT Press, 2009).

29 Shaviro, *Without Criteria, Kant, Whitehead, Deleuze, and Aesthetics*.

30 I borrow the notions of the 'molar' and the 'molecular' here from Deleuze, Gilles and Guattari, Félix, *Anti-Oedipus: Capitalism and Schizophrenia*, trans. Robert Hurley, Mark Seem and Helen R. Lane (London: Continuum, 2004).

31 See also Coleman, *Coding Freedom,*161–84.

32 Ibid., 164.

33 Stengers, Isabelle, 'La guerre des sciences', in *Cosmopolitiques* Vol. 1 (1980): 84.

2

Bend Sinister: Monstrosity and Normative Effect in Computational Practice

Simon Yuill

To learn programming is to embark on years of practice, learning to engage with the unknowable, while battling with complex and sometimes unhelpful theory.

Alex McLean, 'Exclusion in Free Software Culture'[1]

What pleasure does programming give? This daunting summary given by Alex McLean neatly summarizes all that may dissuade someone from taking up the challenge, although for others it may be all the incentive required. Beyond mere masochism, or intellectual bravado, in what forms does pleasure arise in the labour of coding? A common response is that offered under the guise of *elegance*, that there is a certain satisfaction derived from writing an algorithm that performs its task in a particularly effective and clear manner. Such elegance is often postulated as something understood *aesthetically*, as something that is both an intuitive sensuous response and also a matter of cultivated taste. Such is the position put forward in Donald Knuth's *The Art of Computer Programming*. For McLean this pleasure is found by coding in nightclubs. *feedback.pl* is a Perl script by McLean that enables a programmer to perform live improvised music by writing and editing simple algorithms that are executed in realtime – a practice known as *livecoding*. In providing McLean a means of colliding programming and dancing, the xterm[2] and techno, *feedback.pl* unleashes a previously unknown pleasure that situates the arduous labour of coding within the hedonistic monstrations of clubbing: 'with hundreds of people dancing to my Perl, jumping about to my subroutines, whooping as I started up a new script'.[3] As such it challenges many norms of programming practice and societal expectations of what programming is, and reverses the established (non) physicality of the programmer.[4] Even while making this challenge, *feedback.pl* remains a highly skilful exemplification of Knuth's elegant coding which, as Knuth describes it, 'can be an aesthetic experience much like composing poetry or music'.[5] McLean's program itself is brief and conceptually succinct. In an article presenting the project, *Hacking Perl in Nightclubs*, McLean provides a coding example based around a modulo operation on a divisor of four, creating a basic four-four rhythm against which more complex polyrhythms are played, both encapsulating and transcending the quintessence of modern dance music in a few short lines of code:[6]

```
sub bang {
    my $self = shift;
    my $note = 100;
    $note += 50 if $self->{bangs} % 4 == 0;
    $note -= 30 if $self->{bangs} % 3 == 0;
    $note += 60 if $self->{bangs} % 7 == 0;
    beep($note, 40);
    $self->code->[0] = '# note: ' . $note;
    $self->modified;
}
```

But things could turn another way. With minor modifications a program such as *feedback.pl* could direct the programmer's text not to the Perl interpreter but in some other direction such as /dev/dsp, the basic audio output on UNIX and Linux systems. This would result in a situation in which text typed into a computer terminal, rather than composing an algorithm, would instead be directly rendered as raw sound, as used by ap/xxxxx in their performances.[7] The exact same text that in McLean's hands produced a highly danceable minimal techno would now be heard as seemingly random, and sometimes painful, noise.

Another twist might take us in a different direction, like that of Amy Alexander's *extreme whitespace* which injects blank characters into the command line terminal as you type.[8] Also written in Perl and operating according to a similar principle of feedback between terminal input and background process as McLean's program, *extreme whitespace* quickly turns the normally stable environment of the terminal screen into a swirling vortex of characters reminiscent of early video art.[9] These works, of ap/xxxxx and Alexander, also exploit a situational incongruity in their delivery, they exhibit a behaviour we might not expect, in a form we might not predict. They do so, however, through a practice of coding that is less aligned with Knuthian elegance and more closely associated with those used in system hacking and debugging. These bring forth an entirely different set of pleasures, ones that turn from the dexterous to the sinister.

Perhaps, ironically, it is Knuth who best explains such sinister pleasures. In his essay 'The Errors of TEX', he outlines the process of creating what he calls 'torture tests':

> Instead of using a normal, large application to test a software system, I generally get best results by writing a test program that no sane user would ever think of writing. My test programs are intended to break the system, to push it to its extreme limits, to pile complication on complication, in ways that the system programmer could never have consciously anticipated. To prepare such test data, I get into the meanest, nastiest frame of mind that I can manage, and I write the cruellest code I can think of; then I turn around and embed that in even nastier constructions that are almost obscene. The resulting test program is so crazy that I couldn't possibly explain to anybody else what it is supposed to do; nobody else would care! [10]

It is significant how much of the 'torture test' embodies the inverse of Knuth's own paradigms of elegance, in particular that the torture code is so complex as to be unintelligible and impossible to explain. If elegance is dependent upon a certain adherence to or construction of norms, the performance of programming itself is dependent upon the cruel and obscene.[11] These monstrosities are the forms in which the materiality of computational process is convulsed into view, confronting us with disturbing presences that disrupt our normative expectations of how code should operate. For a norm to be effective it must demonstrate, must prove in performance, its ability to transcend such monstrosities. But there is little horror in Knuth's torture, for it seems that it is not so much that he seeks pain but laughter, that moment when physical composure becomes unravelled and erupts into the social.

Throughout his career, Knuth has emphasized that often seemingly pointless but pleasurable activities are an important aspect of how a programmer comes to develop their craft.[12] A recent publication, *Selected Papers on Fun and Games*, documents many of his own such activities. These include gathering photographs of various diamond-shaped road signs, collecting number plates with mathematical puns in them and his first published writing, the fictitious *Potrzebie System of Weights and Measures*, printed by his school journal and then *MAD* magazine in 1957.[13] Indeed, *The Art of Computer Programming* is littered

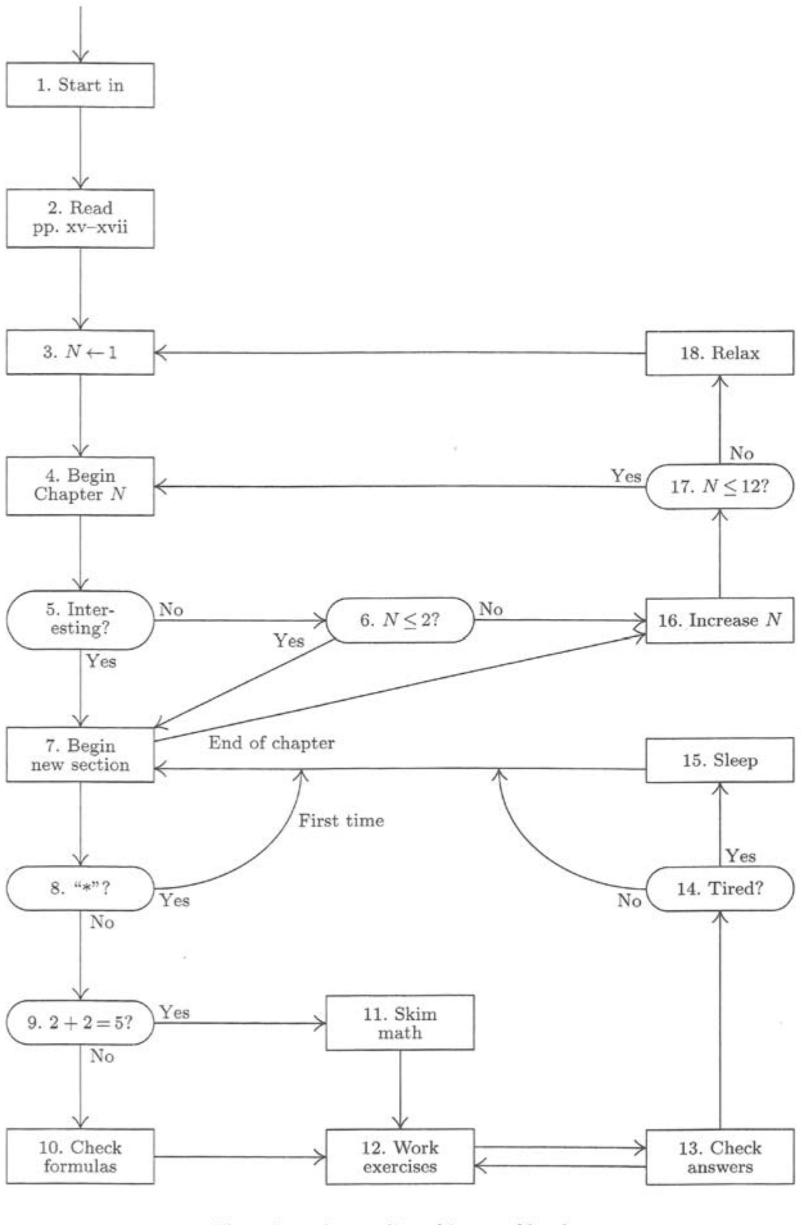


Figure 2.1 Flowchart from 'Procedure for Reading This Set of Books'

Source Knuth, Donald E., *The Art of Computer Programming, Volume 1: Fundamental Algorithms,* 3rd edn, p. xii–xiii. ©1998. Printed and electronically reproduced by permission of Pearson Education, Inc., Upper Saddle River, N J.

with various jokes, games and pranks. The first volume of the series opens with a section entitled 'Procedure for Reading This Set of Books' which is set out in the form of an algorithm accompanied by a flowchart. The reader is admonished to 'follow the steps faithfully', and these include instructions such as:

> **14.** Are you tired? If not, go back to step 7.
> **15.** Go to sleep. Then, wake up, and go back to step 7.[14]

For a book that is regarded by many as one of the defining texts of computing science, and that, for Knuth, constitutes a major life's work (he began writing in 1962 and is yet to complete all seven volumes), this is perhaps not what we might expect. While the style of humour may be particular to Knuth, the fact of humour being so integral to the practice of programming goes deeper than just one writer's authorial manner. Jokes, humour and the absurd have a relation to programming that is not only cultural but also constituent to its very being.

A symptom of professional immaturity

Released by MIT Press in 1978, Roger E. Kaufman's *A FORTRAN Coloring Book* is the first published computing text to use cartoon and comic strip drawings as a pedagogic medium.[15] Its approach has been adopted by a number of other texts on languages such as C, Pascal and Lisp, and, it could be argued, is the archetype to the entire *For Dummies* series and all its numerous imitators.[16] There is, however, something dark within Kaufman's text that sets it apart from the more anodyne humour of these later works. The *For Dummies* books primarily draw upon humour as a means to grease the wheel of cognitive capital, facilitating the ever-recurrent re-skilling (and de-skilling) of the contemporary IT worker. They represent an end-point in the transformation of the use of humour to aid production within the workplace, which has devolved from being a liberal characteristic of privileged workers such as scientists and creatives, as explored in Arthur Koestler's *The Act of Creation* (1964), to being a general form of managerial control known as 'structured fun'.[17] Stylistically, *A FORTRAN Coloring Book* most resembles the world of *Dr. Seuss*, and rather than offering a series of ironic platitudes such as stalk the pages of *For Dummies* books, it engages in a scatological satire that positions the reader in a somewhat Freudian relation to its subject. The opening pages describe the computer as 'like your Mommy's Bureau Drawers', illustrated as a piece of ornate Gothic furniture that looms over the accompanying text. Input/output routines are explained in reference to a part-human, part-bird-like character sat upon the toilet, and arrays are introduced through a short example called FONDLE that traverses the bureau drawers. For all its invocation of the format of children's literature, *A FORTRAN Coloring Book* is not a book for children. Its use of cartoons and

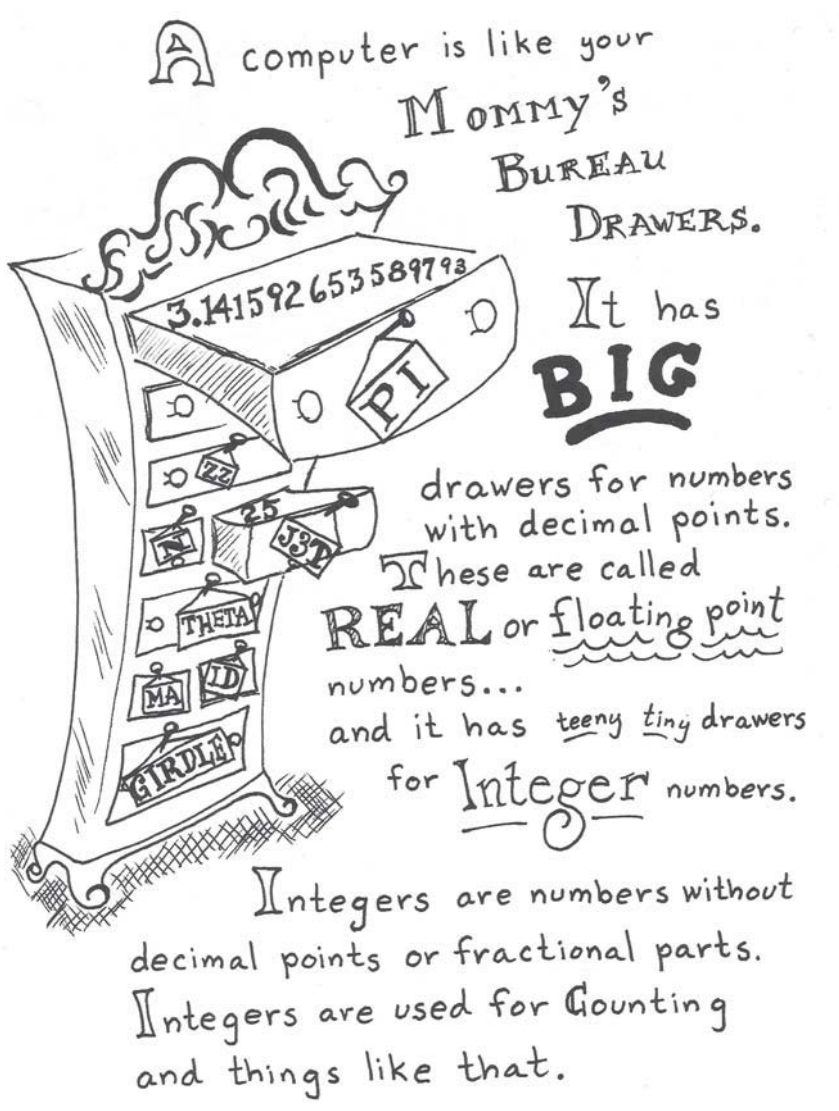

A computer is like your Mommy's Bureau Drawers.
It has BIG drawers for numbers with decimal points.
These are called REAL or floating point numbers...
and it has teeny tiny drawers for Integer numbers.
Integers are numbers without decimal points or fractional parts.
Integers are used for Counting and things like that.
3.14159265358979 3
PI
ZZ
N
25
J3P
THETA
MA
ID
GIRDLE

Figure 2.2 'A Computer is like Your Mommy's Bureau Drawers..'.

Source Kaufman, Roger, *A FORTRAN Coloring Book*, p.1, © 1978 Massachusetts Institute of Technology, by permission of the MIT Press.

the particular bite of its humour have more in common with the unofficial fanzine culture of graduate research communities. These publications, produced on Xerox machine and photo-stencil, were a staple part of environments such as the AI Labs at the Massachusetts Institute of Technology (MIT), and their humour can be seen in the various cartoons that adorned the pages of AI Lab memos, including the early EMACS manuals written by Richard Stallman.[18] Although based at George Washington University, Kaufman gave classes on FORTRAN at MIT, and it was for these that *A FORTRAN Coloring Book* was originally written.

In an essay of 1975, entitled 'How Do We Tell Truths that Might Hurt?' Edsger Dijkstra dismisses the use of anthropomorphisms in computing as 'a symptom of professional immaturity'.[19] While not addressed to Kaufman, such might be the description given to the animate bureau drawers and various hybrid creatures of the *Coloring Book*. From Dijkstra's perspective, this kind of heavily metaphorical presentation of programming merely causes confusion between what programming is and is not, between what it should and should not do. In responding to this critique, McLean has argued that 'metaphor necessarily structures our understanding of computation', and elaborates this through Alan Blackwell's analyses of different metaphors invoked by programmers, in which 'components were … described as actors with beliefs and intentions, being social entities acting as proxies for their developers'.[20] Blackwell's findings echo the encouragement that Knuth gave to students that in writing programming documentation they should find suitable metaphors in which to express how the code works, openly dismissing Dijkstra's position in doing so.[21] But the significance of Kaufman's book is not that it simply provides us with a set of engaging and entertaining metaphors through which we can learn to program. We can read *A FORTRAN Coloring Book* in ways other than those of an instruction manual. Its relation to *Dr. Seuss* is not merely one of shared drawing styles but also one of literary and narrative genres. As a literary work, *A FORTRAN Coloring Book* belongs to the tradition of absurdism and fantasy, of Bakhtinian carnival, that includes the inverse logics of *The Land of Cockaigne*, the hybrid creatures of medieval bestiaries, various works of Lewis Carroll and Douglas R. Hofstadter's *Gödel, Escher, Bach*.[22] As this summary of literary associations suggests, *A FORTRAN Coloring Book* is a work that should be read as an assemblage. In its internal structure it does not conform to a singular genre, it is not purely an instructional manual, nor is it a fully developed satire or, arguably, even that much of a colouring book, but rather as an emergent example of its own genre, it assembles different forms and figurations into itself. The book is also itself an element within a much larger assemblage of other texts, objects and practices. This is an assemblage that includes Kaufman's other writings, programs and engineering projects, but also those of others within the emergence of computer science and the much more circuitous co-evolution of modern humour, logic and computational practices.

Whereas Dijkstra and Knuth contributed to the development of computer science itself, Kaufman has been primarily involved in the application of software to engineering issues. Kaufman came to engineering through work in designing and building theatrical stage equipment, including early experiments in digital control systems.[23] One of his main contributions to computing was KINSYN (KINematic SYNthesis), originally developed in the late 1960s and 1970s, written in FORTRAN and running on an IBM 1130.[24] Developed out of Kaufman's theatre work, KINSYN is regarded as the first visual simulation tool for designing kinematic mechanisms.[25] Early applications of the tool included the design of an orthotic knee brace and, following from this, the development of assistive technologies has been an ongoing area of interest for Kaufman.[26] His other main book, *Introduction to Burmester Theory*, provides a theoretical and mathematical background to kinematic synthesis. It was originally produced in 1973 as a hand-written duplicate and featured cartoons similar to those in *A FORTRAN Coloring Book*.[27]

Like Dijkstra and Knuth, Kaufman began working at a time when much of the standardization and norms of programming languages and practices familiar to current-day programmers were not fully established. He had to implement his own FORTRAN compiler in order to support the calculation of complex numbers required for KINSYN, and in his initial work using SNOBOL (an early symbolic manipulation language developed at Bell Labs) he exposed numerous bugs and errors in the core language implementation.[28]

FORTRAN was developed at IBM in the 1950s, and derived its name from *The IBM Mathematical FORmula TRANslating System*.[29] The emphasis within the original language design was on providing a means of converting mathematical formulae into assembly code instructions. As such it retained many of the features of assembly language programming that explicitly reference the computer memory and hardware, and early versions did not provide the kinds of higher-level abstractions that simplify program structure such as recursion and sub-routines.[30] This also resulted in a lack of standardization across different hardware implementations, with certain routines, such as a DO loop, outputting different results depending on the particular machine they ran on.[31] Writing FORTRAN programs therefore required familiarity with the internal mechanisms of the computer, in order to understand issues such as memory allocation and addressing, output formatting, etc. However, as access to computers was limited during this period (prior to the rise of personal desktop systems), many programmers often had little hands-on experience with the machines they wrote for.[32] A program was typically written by hand on paper, known as a coding form, and typed onto a set of punch-cards which were then fed into the computer. To program, it could be said, was to compose a form of formalized literature directed to an imaginal machine.

The humour of Kaufman's book is endemic to this situation, not merely as a response to its practical frustrations and absurdities, but as a kind of theatrical praxis which mediates between the various manifestations of the human and machinic that constitute its assemblage: the encounter between mathematics, machines and aesthetics. This is based within the theatrical for it depends upon a 'bringing into view' of disparate forms and actions from which a theory of the computational might develop, yet which is not yet fully defined.

First published in 1900, *Le Rire: Essai sur la signification du comique* (Laughter: An Essay on the Meaning of the Comic) has been described as Bergson's only text to attempt a systematic treatment of aesthetics, primarily drawing upon dramatical forms of comedy such as Molière.[33] What is distinctive about Bergson's text is that it situates the humorous, the comedic, within a zone of contagion between the vitality of the human subject and the functional performativity of the automata, between spirit and matter:

> This soul imparts a portion of its winged lightness to the body it animates: the immateriality which thus passes into matter is what is called gracefulness. Matter, however, is obstinate and resists. It draws to itself the ever-alert activity of this higher principle, would fain convert it to its own inertia and cause it to revert to mere automatism. It would … imprint on the whole person such an attitude as to make it appear immersed and absorbed in the materiality of some mechanical occupation instead of ceaselessly renewing its vitality … Where matter thus succeeds in dulling the outward life of the soul, in petrifying its movements and thwarting its gracefulness, it achieves, at the expense of the body, an effect that is comic.[34]

Bergson's attitude towards automatism sits within his broader critique of the mechanical, which, in certain aspects, follows from that of Romantic thinkers such as Herder in positing an opposition between the mechanical and the organic, the material and spiritual.[35] The mechanical, as that which is an artificial assemblage of disparate elements not normally related to one another in 'nature', contrasts with the organic as that which is constituted from innately related elements forming an integrated whole in which each performs its 'naturally' given role or, in Bergson's conception, the integrity of their common duration.[36] The machinic has materiality and motion, but no spirit or grace, for its motion derives from the spatialization of time as number. This takes the mechanical as comprising both physical machines and institutional forms – the examples in *Le Rire* include both the operation of machines and the formalistic behaviour of bureaucrats. Bergson contends that the comic lies in the exposure of a repetition in behaviour which, like mathematics, exists outside the experience of time, contradicting the accrual of difference embodied in his notion of duration. However, while he touches upon the role of repetition in poetry, he does not make a clear distinction between repetition that is comical and that of, say, dance or music, which may similarly contradict our sense of time as duration yet be elegant, joyous or grave.[37] As his examples demonstrate, it is not the fact of repetition in itself from which the comical arises, but rather, that the repetition appears incongruous in regard to existing social norms. Humour here performs a normative role, it 'corrects men's manners' and differentiates society according to that which is desirable and undesirable within a given social order:

> Laughter must be something of this kind, a sort of *social gesture*. By the fear which it inspires, it restrains eccentricity, keeps constantly awake and in mutual contact certain activities of a secondary order which might retire into their shell and go to sleep, and, in short, softens down whatever the surface of the social body may retain of mechanical inelasticity. Laughter, then, does not belong to the province of esthetics alone, since unconsciously (and even immorally in many particular instances) it pursues a utilitarian aim of general improvement.[38]

Humour relates the aesthetic to the moral. That which becomes comical is that which fails the judgement of 'good sense' – Bergson's *Le Bon Sens* addresses issues similar to *Le Rire*.[39] This 'good sense' includes examples that today's reader might consider merely prejudicial, such as the discussions about the hunchback and Negro as objects of laughter. Humour and its morals are historically and culturally situated, and similar examples are to be found in Schopenhauer's treatment of humour in *The World as Will and Idea* (1819) and Lewis Carroll's *Symbolic Logic* (1896).[40] As an instrument of social differentiation, humour delineates the terrain of discrimination from which dominant groups and classes operate. It may also, however, facilitate practices through which various alternate assemblies seek to constitute themselves against that.

Eric Raymond's *The Jargon File* is a compendium of slang, peppered with various jokes and re-instrumentalizations of language through which the 'old school' hacker culture, in which Kaufman worked, identifies itself.[41] In her study of hacker culture, Gabriella Coleman defines humour as integral to hacking practice, not only as a form of social differentiation, but also as a modus operandi. Coleman argues that the structure of the hack, as the bringing together of unrelated elements so as to produce an unexpected functionality, parallels that of the joke, citing Mary Douglas's definitions of joking as a practice that combines 'disparate elements in such a way that one accepted pattern is challenged by the appearance of another'.[42] She illustrates this with an example of a succinct Perl hack that, in its attentiveness to the aesthetics of the language, is analogous to those of McLean's *feedback.pl* and Alexander's *extreme whitespace*:

```
#count the number of stars in the sky
$cnt = $sky =~ tr/*/*/;
```

The code counts the number of asterisks (stars) within a piece of text held by the variable named $sky compressing what might normally be six lines of code into one by exploiting 'certain side effects found in the constructs of the Perl language', and thereby creating a kind of poetical play within the source code of a program.[43]

In his theory of the joke as a 'diagram of innovative action' Paolo Virno describes the joke as an empirical use of language that tests the contingency of normative behaviour and grammatical constructs.[44] The joke is a playful use of language that performs unexpected twists of meaning, short-circuits of logic and, in so doing, disrupts both linguistic and social norms. In developing this theory, Virno adopts a set of terms from Aristotle's *Nicomachean Ethics*:

> a.) *phrónesis*, or practical know-how; b.) *orthós logos*, the discourse that enunciates the correct norm according to which the action in one single case takes shape; c.) the perception of *kairós*, of the proper moment for performing an action; d.) *éndoxa*, that is, the opinions prevalent from time to time within a community of speakers.[45]

In particular, the joke applies innovative, or unexpected, forms of know-how, *phrónesis*, in order to test the viability of given *éndoxa*, and thereby expose the contingency of 'all situations' and 'the way in which these situations are to be dealt with'.[46] As such, Virno's theory is entirely the opposite of Bergson's, in which the joke serves to establish and reinforce prevailing *éndoxa*.

Among his personal engineering projects, Kaufman demonstrates a hacker-inventiveness in his *PantsPutterOnner* and *ShirtPutterOnner* devices which, at first glance, have the appearance of Heath Robinson or Rube Goldberg contraptions, made, in one case, from elements that look to be adapted from household plumbing. These are a 'pair of dressing machines … designed and built … for a student born without arms'.[47] Their construction epitomizes the elegance of a hack that, taking existing materials and given existing norms, makes possible that which might otherwise not be. Susan Leigh Star has questioned the way in which technologies are brought to address such needs, as though it simply 'were a matter of expanding the exhaustive search for "special needs" until they are all tailored or customized'.[48] She calls instead for a questioning of the 'distribution of the conventional', asking, 'What is the phenomenology of encounters with conventions and standardized forms, as well as with new technologies?'[49] Unlike cosmetic prostheses, however, Kaufman's devices do not seek to standardize the body in conformance with societal norms (such as we would impose if we were to give the student prosthetic arms and insist he dress 'like everyone else') but rather accept and accompany the given physicality of their user. Seemingly arising from the joke and the very forms of that which is derogated in Bergson's 'utilitarian aim of general improvement', the automata and the 'deformed' body, the dressing machines instead demonstrate that even the most mundane of activities, that of getting oneself dressed, can bring into action numerous forms of *phrónesis* entirely outside existing *éndoxa*.

The *orthós logos* announced by Bergson, that it is laughable for a living body to incorporate the machinic (the becoming-machine), mirrors that of Dijkstra, that it is immature for the machinic to be given human qualities (the becoming-human).[50] What emerges between the elements assembled across Kaufman's practice, however, between the dressing machines and the colouring book, is neither simply mecha-morphic nor anthropomorphic, but rather that which 'affords opportunity for realising that an accepted pattern has no necessity'.[51] The hack/joke replicates the structure of the machinic. Each constitutes an assemblage of elements that are not 'normally' brought into relation – whether in terms of societal norms, linguistic standards or conventions of the 'natural'. Kaufman's work does not succumb to the criticisms of Bergson or Dijkstra, however, but rather exposes their contingency. Virno ascribes the 'violence' of human language to its potential to negate being, to say that a person is a not-human. In his critique of Hegelian dialectic, Bergson dismisses the category of the negative as the construction of a 'false problem' that constructs criteria that cannot exist, and yet laughter in *Le Rire* is itself the performance of such 'violent' negation.[52] Here, laughter responds to and identifies the not-human within the human, just as Knuth's torture test identifies the not-computational within his own software code. For Virno, however, humour can move beyond such first-order negations to not only expose the contingency of such patterns but also, as in Kaufman, to become the negation of negation.[53]

Kaufman constructs a comedic theatre that operates through a 'bringing into view' of all that which is positively regarded (that which constitutes 'theory', the thing we choose to contemplate) and that which is negated (the not-human, the not-computational) and presents them as an ad hoc assemblage that is prior to any such selection or normative structure. In Greek tragedy the body of the dead hero would be revealed upon the *ekkyklêma*, a piece of machinery wheeled onto the stage, exposing a truth that might otherwise be hidden.[54] In comedies the body would spring to life, or something entirely unexpected would appear. Kaufman's play brings on stage the machinery of computation and its concomitant desires and ambiguities. His characters perform the hybrid agency of human and non-human actors, the 'promises of monsters' of which Donna Haraway wrote: 'Their boundaries materialize in social interaction among humans and non-humans, including the machines and other instruments that mediate exchanges at crucial interfaces and that function as delegates for other actors' functions and purposes'.[55]

The promise is not entirely made good in Kaufman, however, for there are conflicting tendencies within his satire that echo the 'unequal structuring' between objects, science and nature whose problematic Haraway maps. Kaufman delights in the base corporeality of the computational in a form of Bakhtinian grotesque, yet, at the same time projects ambivalent anxieties through the construction of the computer as the mother-bureau hybrid, both desired and feared. This Oedipal configuration invokes the female-monster figures of eighteenth-century satire, such as Swift's Goddess Criticism and Spenser's Errour. These, as Susan Gubar has argued, allegorize what their authors considered to be forms of dangerous and unruly writing conflated with uncontrollable natural processes of endless eating, defecating and childbirth.[56]

This suggests a different reading of the anthropomorphic in regard to the computational, one that expresses both that which gives a human form to a non-human object but also that which shapes the human. The 'hurtful truth' pronounced by Dijkstra was directed against what he considered to be the false claims of John von Neumann that human brains were a form of computing machine that a computer could mimic and the ideological ambitions of AI.[57] Dijkstra's critique can be understood as an accompaniment to the distrust of machinery that haunts his other writings, and in response to which he enacted his own anthropomorphism. Dijkstra performed computation itself as an inherently human action, insisting on working out programs in long-hand writing rather than on a machine (long after coding forms became obsolete) and arguing that the true computer for which the programmer wrote was not the compromised physicality of hardware but the pure intellectual 'machine' of the programming manual:

> Eventually, there are two 'machines'. On the one hand there is the physical machine that is installed in the computer room, can go wrong, requires power, air conditioning, and maintenance and is shown to visitors. On the other hand there is the abstract machine as defined in the manual, the 'thinkable' machine for which the programmer programs and with respect to which the question of program correctness is settled.... Originally I viewed it as the function of the abstract machine to provide a truthful picture of the physical reality. Later, however, I learned to consider the abstract machine as the 'true' one, because that is the only one we can 'think' ...[58]

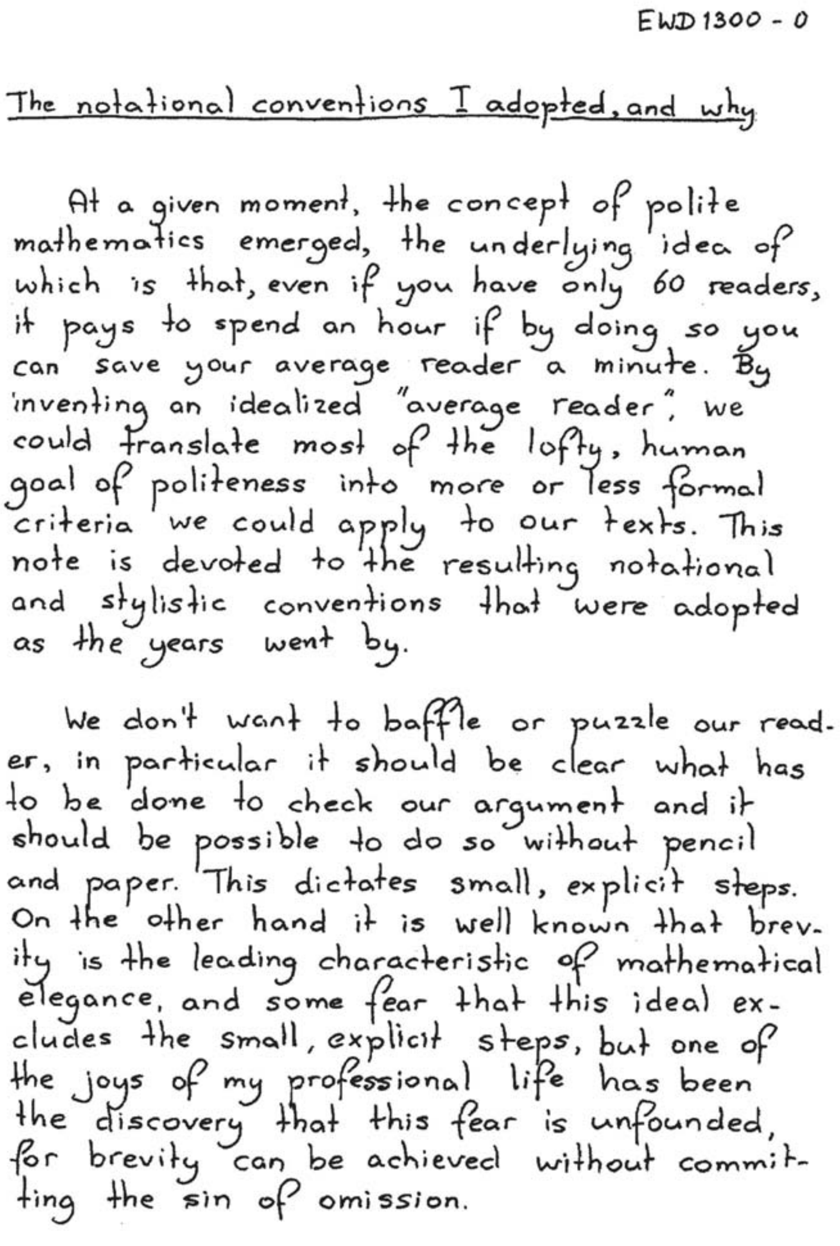
There is an irony in Dijkstra's practice for it is almost as though he were unable to escape the *mise-en-scène* of Turing's description of the problem of computation as that of a human 'computer' sitting at a desk writing and erasing marks on an endless strip of paper, not knowing when, or if, he can stop.[59] There is a double irony in that Dijkstra pioneered the interrupt mechanism, an element relating external hardware, such as keyboards, to the internal processing system. The interrupt enables a programmer to write and run code interactively on a computer, a key innovation that rendered the coding form obsolete, and without which McLean's *feedback.pl* would not be possible.[60] Dijkstra's distrust also echoes that of many mathematicians towards computer-demonstrated proofs, such as evidenced, famously, in the lukewarm response to Appel and Haken's proof for the Four Colour Theorem (1976). The theorem seeks to determine whether or not all the countries on a map can be coloured in using only four distinct colours and ensuring that no two neighbouring countries are coloured the same. This was the first major theorem to be proven using software assistance having eluded purely human analysis since it was originally conjectured in 1852, yet its solution was not celebrated.[61] The negative response of the mathematics community is epitomized in the words of one critic, Ian Stewart, who argued that this approach did not explain *why* the proof was correct:

> This is partly because the proof is so long that it is hard to grasp (including the computer calculations, impossible!), but mostly because it is so apparently structureless. The answer appears as a kind of monstrous coincidence. Why is there an unavoidable set of reducible configurations? The best answer at the present time is: there just is. The proof: here it is, see for yourself. The mathematician's search for hidden structure, his pattern-binding urge, is frustrated.[62]

The Appel-Haken proof confounded the aesthetics of mathematicians for the set of 1,482 different map configurations required to verify it could not be grasped by human imagination.[63] It was not succinct. It was not elegant. In combining human and mechanical means it transgressed the prohibition against crossing between different disciplines such as that between arithmetic and geometry, the *metabasis ex allo genos*, as established in Aristotle's *Posterior Analytics*.[64] The proof exuded an excess of computational materiality, it had contagion, it could not be given human shape.

Dijkstra's long-hand computations can be fully understood as anthropomorphic in the dual sense of that which gives human shape to something, and as that which shapes the human. Elsewhere in the essay critiquing anthropomorphism Dijkstra argues: 'The tools we use have a profound (and devious!) influence on our thinking habits, and, therefore, on our thinking abilities'.[65]

The choice between computational machine on the one hand and pen and paper on the other is, for Dijkstra, a conscious choice of how the programmer himself is shaped.[66] The expression of such a choice is also the projection of a particular becoming-human onto the tool itself, and therefore a form of anthropomorphism. As Paul de Man writes, the anthropomorphic is that which seeks 'to reconcile the pleasures of the mind with those of the senses and to unite aesthetics with epistemology'.[67] Barany and MacKenzie argue that the persistence of chalk and blackboard within mathematical practice, long after digital displays and PowerPoint have dominated all other areas of academic presentation, is that the physical writing of the equations constitutes a form of performance through which mathematical proofs are given material validity.[68] It is as though the equation must be written out, rather than simply shown on a slide, in order for it to come into being as an object that we can perceive (aesthetics) and comprehend (epistemology). When aesthetics and epistemology fail to coincide the effect may either be monstrous or comic.



EWD1300 - 0

The notational conventions I adopted, and why

At a given moment, the concept of polite mathematics emerged, the underlying idea of which is that, even if you have only 60 readers, it pays to spend an hour if by doing so you can save your average reader a minute. By inventing an idealized "average reader", we could translate most of the lofty, human goal of politeness into more or less formal criteria we could apply to our texts. This note is devoted to the resulting notational and stylistic conventions that were adopted as the years went by.

We don't want to baffle or puzzle our reader, in particular it should be clear what has to be done to check our argument and it should be possible to do so without pencil and paper. This dictates small, explicit steps. On the other hand it is well known that brevity is the leading characteristic of mathematical elegance, and some fear that this ideal excludes the small, explicit steps, but one of the joys of my professional life has been the discovery that this fear is unfounded, for brevity can be achieved without committing the sin of omission.

Figure 2.3 One of Prof. Dr Edsger W. Dijkstra's handwritten papers, discussing 'polite' notation, EWD1300

Source Full text: http://www.cs.utexas.edu/users/EWD/ewd13xx/EWD1300.PDF

Selfish computers of happiness

Knuth's 'Computer Programming as an Art' was originally presented as his acceptance speech for the 1974 Turing Award.[69] It takes the form of a survey of different discussions about art and aesthetics in relation to computation and number with references that range from a definition of logic by thirteenth-century philosopher Duns Scotus to a nineteenth-century manual on accountancy. Within this there are two philosophers whom he quotes at length and gives particular prominence to: Jeremy Bentham and John Stuart Mill.[70] From Bentham he derives the concept of utility as the best measure of good and bad: 'There is no taste which deserves the epithet good, unless it be the taste for such employments which, to the pleasure actually produced by them, conjoin some contingent or future utility…'[71]

Bentham's concept of the good as that which is both measurable and pleasurable, and given algorithmic expression through his felicific calculus, underlies and gives a moral dimension to Knuth's concept of programming elegance and its more pragmatic expression, explored at some length in *The Art of Computer Programming* and other texts, of seeking to quantitatively measure the scale and efficiency of a given algorithm in terms of its running time and use of resources.[72] From Mill he derives the argument that art and science are complementary, that all disciplines have both an art and a science to them, with art supplying the capacity to judge that knowledge produced by science which has greatest value to practice: 'Art … brings together from parts of the field of science most remote from one another, the truths relating to the production of the different and heterogeneous conditions necessary to each effect which the exigencies of practical life require'.[73]

Here, art is that which defines the *conditions* under which knowledge becomes productive. For Knuth these conditions are both *economic*, the effective use of resources, and *aesthetic*, a sense for that which is harmonious and 'good'. Bentham and Mill help define what may be described as Knuth's ethics of production, the behaviours and moral values under which programming is practised. The realization of this in elegant code gives material shape to the larger project of Literate Programming which Knuth has advocated throughout his career and created software tools to facilitate. Just as these ethics draw upon the ideas of English liberal philosophers, Bentham and Mill, the practice of Literate Programming also adopts the aesthetic medium most closely related to the expression of liberal thinking, the essay: 'The practitioner of literate programming can be regarded as an essayist, whose main concern is with exposition and excellence of style'.[74]

For the Earl of Shaftesbury the essay was the ideal vehicle through which to define and disseminate the new subjectivity of the self-possessing liberal individual.[75] This was given form through Shaftesbury's concept of *politeness* as a mode of cultural and political action. Acquired through appropriate education and demonstrated through exhibiting appropriate taste, politeness offered a means of acquiring authority within the transactions of daily life and commerce that did not derive from the institutional power of the monarchy or the Church whose influence had fallen in the wake of the English 1688 Revolution. Depending as it did upon access to a privileged means of education and consumption, the practice of politeness retained social differentiations while being able to operate across all contexts, assuring both liberty and distinction to those who had it. Politeness was both a literary and theatrical practice for it was disseminated through particular modes of writing, and applied as a kind of extemporary performance on the social stage. This gave voice to a moral philosophy that was conversational and concrete rather than speculative and overly abstract, and whose aesthetic was its means of appeal: 'The author wrote in an agreeable English. He punctuated the discourse with humour. He preserved moral scale, eschewing, at one end of the spectrum, excess detail and elaboration and, on the other, mysterious depth and abstraction'.[76]

While politeness gave qualitative polish to the emerging discourse of English liberalism, it was conjoined with another mode of writing that gave it an effect of quantitative rigour, that of mercantile and scientific numerical writing. This is described by Mary Poovey as a *gestural mathematics*, the use of mathematical language to describe philosophical concepts and thereby suggest a correlation between the methods of mathematical proof and philosophical argument.[77] Rather than defining the aesthetic as an opening to pure sensuality (as the Romantics would later do), the mathematical precedes and shapes the aesthetic as defined in this era, for it is from the potential of mathematical thought to consider things in the abstract that the notion of the disinterested contemplation of the aesthetic derives. Similarly, the concept of the beautiful derives from analogy to the harmony of mathematical proportions. This abstract basis of aesthetic judgement appears to run counter to the tangibility of discourse promoted by Shaftesbury. For Shaftesbury, however, this is resolved in terms of class. The aesthetic is not that which is abstract from the world but rather from labour, for those who work with their hands, absorbed in materiality through their labour, are unable to think in terms of either the abstract or the disinterested; they are, in a word, unaesthetic.[78] Aesthetics here is a moral-political practice, it makes the abstract concrete and gives it *value*, for it seeks to instil behaviour that is proportionate and 'true'.

Humour is integral to the realization of this. In 'Sensus Communis; an Essay on the Freedom of Wit and Humour' (1709), Shaftesbury argues that irony and satire (what he calls 'raillery') are an ideal means of testing the logic and substance of debate:

> They may perhaps be Monsters, and not Divinitys, or Sacred Truths, which are kept thus choicely, in some dark Corner of our Minds: The Specters may impose on us, whilst we refuse to turn 'em every way, and view their Shapes and Complexions in every light. For that which can be shewn only in a certain Light, is questionable. Truth, 'tis suppos'd, may bear all Lights: and one of those principal Lights or natural Mediums, by which Things are to be view'd, in order to a thorow Recognition, is Ridicule it-self, or that Manner of Proof by which we discern whatever is liable to just Raillery in any Subject.[79]

In subjecting ideas to the test of ridicule, humour acts as an 'instrument of reason' exposing that which claims to be proportionate and true yet which rests upon a logic that is deformed and ugly.[80] As in Knuth's torture test, raillery debugs theory.

For the second edition of *Characteristicks* of 1714 Shaftesbury commissioned a series of engraved iconographies to illustrate its subject matter from the artist Simon Gribelin and worked closely with the printer, John Darby Jr, to produce the publication to exact standards ensuring that no ink showed through the pages as had happened with the first edition.[81] Dismayed by the poor quality of the electronic typesetting of the second edition of *The Art of Computer Programming*, Knuth set about writing his own software to typeset his books. Intended as a short sabbatical project, the work stretched to more than ten years and culminated in TEX, now one of the most widely used digital publishing tools.[82] The first book to be fully typeset in TEX was *Concrete Mathematics* (1989). Written in collaboration with Ronald L. Graham and Oren Patashnik, the book grew out of a course taught at Stanford on the mathematical analysis of algorithms, started by Knuth in 1970 and first set out in the opening chapter of *The Art of Computer Programming*. A new font, called AMS Euler, was specially commissioned from designer Hermann Zapf to typeset the mathematics in the book. Echoing Dijkstra's preference for writing on paper, the font was designed to 'capture the flavour of mathematics as it might be written by a mathematician with excellent handwriting. A handwritten rather than mechanical style is appropriate because people generally create mathematics with pen, pencil, or chalk'.[83]

This explicit evocation of the handwritten can be read in relation to the authors' statement that the book is 'a kind of manifesto about our favourite way to do mathematics'.[84] A manifesto that is also expressed in the book's title, *Concrete Mathematics*, emphasizing the practical applicability of what it taught and consciously rejecting the Abstract Mathematics then in vogue in US universities and the New Math educational systems.[85] The book also evokes another entirely different practice of inscription. Running throughout the margins of the book are various jokes and comments gathered from students on the course that the authors call 'graffiti':

> Some of these marginal markings are merely corny, some are profound; some of them warn about ambiguities or obscurities, others are typical comments made by wise guys in the back row; some are positive, some are negative, some are zero. But they all are real indications of feelings that should make the text material easier to assimilate.[86]

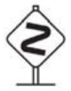
This might seem a little idiosyncratic and self-indulgent, yet, in the light of the role of humour in Shaftesbury's doctrine of politeness and the authors' own explanation, it can be understood as integral to Knuth's ethics of production. It is as though the authors, in a display of polite self-deprecation, wish to demonstrate that the subject matter has been subject to the 'raillery' of the classroom. It also points towards a possible awareness on the authors' part of the role of humour in establishing and cultivating, what might be called, the normative pleasures of their practice. This connects to Bentham, in relating the intrinsic 'good' within a practice to the investment of pleasure and also to Knuth's support for recreational mathematics and toy problems as a means for developing that practice through disinterested, pleasurable activities. As Stanley Grean puts it in his study of Shaftesbury, humour here is 'both a method and a state of mind'.[87] Whereas, however, the Shaftesburian text gestures towards the mathematical so as to construct a sense of legitimizing order beneath its discourse, the graffiti of *Concrete Mathematics* and the jokes that punctuate *The Art of Computer Programming* are a form 'gestural comedics' whose role is not that they entertain or make us laugh, but rather that they correlate the subject matter of the texts to a particular temperament and sensitivity. It is through the gesture of the comedic that the graffiti construct and reinforce a particular *éndoxa*, 'the opinions prevalent from time to time within a community of speakers'. Literate Programming, therefore, is not simply a method for writing programs, but also of making programmers, of shaping subjectivities. Humour constructs a kind of class ethics. As though to paraphrase Shaftesbury: 'For without Wit and Humour, software can hardly have its programmers or be programmed'.[88]

Both Knuth and Kaufman reflect upon the teaching situation through their humour, but whereas for Kaufman this negotiates the jointly corporeal and imaginal relation between human and machine, in Knuth it negotiates the roles of student and tutor. The pedagogical context for Knuth is not simply one phase within that process but rather intrinsic to programming itself. Knuth describes the act of writing programs as like teaching somebody else to do a task. The programmer teaches the machine. The machine, however, also teaches the programmer through the way it enforces greater precision on the 'tutor' than a purely human teaching scenario might require.[89] The teaching situation was a recursive one, and this recursion gives rise to a new social formation: 'I thought about how it would be to live with such a machine and with the new tools that I was creating – sort of like living in a new subculture'.[90]

Like the Shaftesburian essay, Literate Programming teaches one how to behave in a new social order. Knuth's humour sits within the genealogy of polite liberal humour that stretches from Shaftesbury to Bergson for it serves to inform the normative conditions under which such a social and cultural order comes into being, a 'social gesture' that 'pursues a utilitarian aim of general improvement'. It does so through the construction of an ethics of production which is also a 'class' ethic, that, as explored in the work on Gabriel Tarde, links the schoolroom to the economic structure.[91] Elegant code, in this context, articulates the identity of the programmer within a particular professional class, equivalent to the English liberal conception of the 'order of ranks' and the French professional *cadres*.

While Shaftesbury was a leading influence upon and advocate of Whiggism, he was in certain respects reactionary in regard to his times and the later radical liberalism of Bentham. Looking back to the older concepts of virtuous republicanism of the pre-revolutionary writers he criticized those who embraced the explicitly quantitative values of emerging mercantilist society as 'selfish Computers of Happiness'.[92] Despite the references to Bentham, Knuth does not give over to a fully economic utilitarian conception of computing under the demands of *homo* *oeconomicus*, perhaps owing something to a belief in the principle of academic work as a shared public good. The full economic and political consequences of a return to Bentham in twentieth-century computing do not arise from Knuth's work. It is within von Neumann and Oskar Morgenstern's *Theory of Games and Economic Behaviour* (1944) that the selfish computation of happiness as economic utility finds a new form. While drawing on the more rigorous mathematical models of Daniel Bernoulli in implementing their system, they directly invoke Bentham's historical and moral arguments in justifying their approach.[93]

The instrument of humour has a more explicitly 'ideological' sense, however, in how *Concrete Mathematics* and *The Art of Computer Programming* position themselves in regard to other manifestos and philosophies within twentieth-century mathematical practice. Across the pages of *The Art of Computer Programming*, there lurks a spectre, making itself known only through the presence of a sinister sign:



It appears on the margins of the text to warn the reader of passages which may be of particular difficulty and require greater attention or caution. The sign takes its form from the roadsign symbol for a dangerous bend. While this may appear to be derived from Knuth's hobby of collecting diamond-shaped roadsigns, the origins of its use lie not within the US highway code but elsewhere within mathematics. The symbol was first introduced in *Éléments de mathématique* (Elements of Mathematics) by Nicolas Bourbaki, the first volume of which was published in 1939. While Knuth's writing style may follow the Shaftesburian tradition, the form and ambition of *The Art of Computer Programming* most clearly follows that of *Éléments de mathématique*. Both are multi-volume works which seek to provide a near definitive account of their discipline. Both, as a series, remain unfinished. Both works gradually build from basic first principles, set theory in Bourbaki, data structures and algorithms in Knuth, towards more complex domains. Distinctively, and unusually for mathematical and computing texts, both combine their theoretical expositions with historical notes outlining prior developments. And, prior to Knuth, it is in Bourbaki that we find the first acknowledgement that recreational mathematics has been a significant source of innovation and discovery.[94] Similarly, both Knuth and Bourbaki's works attempted to standardize mathematical notation and introduced their own symbols. Bourbaki introducing not only the dangerous bend symbol but also, more significantly, the sign for an empty set, $\varnothing$, which became a fundamental building block to their theory of number.[95] Knuth introduced the up-arrow symbol, ↑, for notating large integers, and sets out in the introduction to *The Art of Computer Programming* distinct uses of notation that separate the description of an algorithm from its mathematical analysis.[96] Typically for Knuth, the introduction of notation is not without humour, and at one point in his discussion he declares that '[t]he notations we are using in this section are a little undignified. … Our notations are almost universally used in recreational mathematics (and some crank literature!) and they are rapidly coming into wider use'.[97]

Despite these similarities Bourbaki is not cited in Knuth's major works. The formalized structural text of Bourbaki is the antithesis of the convivial elegance of Knuth. More fundamentally, Bourbaki was a key influence on the emergence of New Math teaching and the dominance of Abstract Mathematics against which *Concrete Mathematics* positioned itself.

Not only does the distinction between Knuth and Bourbaki relate to that of two opposing ways of doing mathematics but they each embody two opposing models of the construction of subjectivity and the role of humour within that. With Knuth humour coheres the subject, drawing upon the practices of subject formation from the rational liberal tradition. A private individuated 'self' that comes into being through its display of established normative patterns that are, in part, demonstrated through the humour that it performs. With Bourbaki, conversely, humour performs the very undoing of such a 'self'. There has never been a single, coherent Nicolas Bourbaki as a self who authored a series of mathematical texts, rather Nicolas Bourbaki is a syndicate of mathematicians. The name itself was, according to one account, originally given to a visiting speaker who would present the annual lecture to first-year students of the *École Normale Supérieure* in Paris. The lecture was a fiction, a prank, performed by an actor, the name chosen at random.[98] When a group of mathematicians connected with the *École* began working on a project to provide a comprehensive theory of contemporary mathematics, they adopted the name as their *nom de plume*. The group included, among many others, Henri Cartan, Jean Dieudonné and André Weil. Weil's description of their collectivized authorship presents a kind of literary practice that could not be further removed from the explicit personalization within Shaftesburian essay technique:

> For each topic, a writer was designated after a preliminary report and group discussion. This writer provided a first draft which the group would read and discuss again, modifying it to varying degrees or even, as happened more than

once, rejecting it out of hand. Another writer would be designated to come up with a second draft, following the directives of the group – which of course were not always heeded; and so on.[99]

The prank was taken on by the mathematics community, many of whom continue to refer to Bourbaki as a singular individual.[100] The use of a syndicated identity was later adopted by other mathematicians, such as the Situationist-style maths fanzine *Manifold*, literary collectives such as OuLiPo, which included mathematicians and sought to base a new practice of writing on Bourbaki's axiomatic system, and appears in multi-use names such as Luther Blissett and Karen Eliot.[101] In the 1940s Bourbaki's attempt to join the American Mathematical Society (AMS) as an individual member was rejected by the secretary Ralph Boas who published an article denouncing Bourbaki as a group. In a manner prefiguring the deliberately provoked feuds of Stewart Home and the Neoists, Bourbaki responded with a letter refuting the allegations and claiming that Boas was a fictitious name under which a group of US mathematicians published the *Mathematical Reviews*.[102] Boas, it so happens, was also on the committee who commissioned Zapf to design the AMS Euler font, a font which in its evocation of a handwritten style foregrounds the idea of mathematics as the pursuit of self-contained subjects who are coterminous with their physical bodies. In contrast to Bourbaki's use of a common syndicated identity to recognize the collective endeavour of their mathematics, the authors of *Concrete Mathematics* explicitly named individual contributors.[103]

While Dijkstra's insistence on writing by hand might demonstrate a similar commitment to the coterminous self embodied in AMS Euler, his specific choices in writing machinery indicate a more complex, multiple performance of the self within his work. Among Dijkstra's self-published writings there are some that are typed out on letterheaded paper rather than handwritten. The letterhead is for Mathematics Inc., an international corporation, the chairman of which is none other than Dijkstra himself. This is not Dijkstra as scientist and mathematician but rather his own nefarious alter ego. The letters are included within the indexing system Dijkstra applied to his other scientific papers and included in an anthology of his work published by Springer in 1982, thereby giving them equal importance to his mathematical and computing work.[104] These are not scientific notes, however, but a satirical parody of modern management techniques and the commercial exploitation of mathematical knowledge. Central to Mathematics Inc. strategy is the ruthless pursuit of legal ownership and patent rights over mathematical theorems. Written in the wake of the establishment in 1967 of the World Intellectual Property Organization (WIPO), they can be seen as part of the longer struggle over Intellectual Property in academia and the later rise of copyleft and Free Software. In this regard, they have something in common with the authorial devices of Luther Blissett and Karen Eliot. Dijkstra's split self is not, however, the embracing of the syndicated identity of Bourbaki or Blissett but more of a Scriblerian attack against the influence of commercialization on academic research.[105] In one Mathematics Inc. piece, which is prescient of current directions in the corporatization of the university, the company announces a new product, the Mathematical Articles Evaluation System (MAES®), designed to automate the grading of mathematical papers and thereby undermine the autonomy of academic practice: 'the wholesale introduction of MAES® will teach the reactionary bastards! At last their private, hobbyist norms will evaporate, for MAES® will force them to adopt the standards of the mathematical industry'.[106]

MAES embodies an entirely different literary form, neither the privileged personal expression of the Shaftesburian essay, nor the syndicalized commonality of Bourbaki, but rather the corporatized, branded voice of *homo oeconomicus*. The algorithmic authorship executed by MAES restructures all thought under a singular legal regime rather than the debates of mathematical discourse.

Even as a joke isn't it evidently mathematics?

One of Dijkstra's main contributions to computing was his involvement in the design and implementation of the ALGOL programming language. ALGOL (from *ALGOrithmic Language*) was designed as a universal programming language which would not only work uniformly across different types of hardware but also provide a clear written presentation of an algorithm suitable for printing in scientific reports and journals. As such, it was intended both as a response to the inefficiencies and limitations of FORTRAN and, like Knuth's later development of Literate Programming, to support the dissemination of computing knowledge and pedagogics.[107] Whereas Literate Programming would adopt the use of an essay form wrapped around the programming code, in ALGOL the syntax and typographic conventions of the language itself were intended to provide sufficient clarity so that an algorithm presented in ALGOL could be read as a self-contained expression. This foregrounds the structure of the algorithm itself rather than the machine-specific descriptions of memory registers and allocations that were typical of FORTRAN code. ALGOL describes what the algorithm will do, while FORTRAN describes what the machine will do. In this way algorithms could be published in a written form analogous to that of the publication of a mathematical proof, rather than, as with FORTRAN, something akin to engineering notes. Unlike previous programming languages, which were often devised to suit a particular task, ALGOL was designed as a set of basic generalized axioms from which more complex statements could be derived. As such, it follows the work of Hilbert's project, as outlined in 'The Foundations of Mathematics' (1927), of defining mathematical practice in terms of how it can be expressed through a rigorous formalized language:

> For this formula game is carried out according to certain definite rules, in which the *technique of our thinking* is expressed. … The fundamental idea of my proof theory is none other than to describe the activity of our understanding, to make a protocol of the rules according to which our thinking actually proceeds.[108]

The emphasis upon formalization is reinforced in a later comment by Dijkstra in which he argues in favour of the phrase 'programming notation' rather than 'programming language':

> The introduction of the term 'language' in connection with notation techniques for programs has been a mixed blessing. On the one hand it has been very helpful in as far as existing linguistic theory now provided a natural framework and an established terminology ('grammar', 'syntax', 'semantics', etc.) for discussion. On the other hand we must observe that the analogy with (now so-called!) 'natural languages' has been very misleading, because natural languages, non-formalized as they are, derive both their weakness and their power from their vagueness and imprecision.[109]

Although Dijkstra was as strong an advocate of elegance as Knuth, this comment indicates something of a distinction in how each understood this.[110] While both might place emphasis upon the precise, efficient expression of an idea in code, for Knuth this has a rhetorical dimension in that code, as essay, should aim to be persuasive in expression and display an appropriate conduct on the part of the programmer – which can be contrasted with the 'obscene' and 'crazy' code of his torture test programs. For Dijkstra elegance lies more in an irrefutably self-evident correctness, for truly elegant code would not require commentary nor debugging.[111] For Knuth elegance is the start of a conversation, for Dijkstra it is the conclusion. While Knuth cites Bentham and Mill in defence of his ideas, the opening pages of the *Report on the Algorithmic Language ALGOL 60* quote the *Tractatus Logico-Philosophicus* (1921) of Wittgenstein: 'Was sich überhaupt sagen läßt, läßt sich klar sagen; und wovon man nicht reden kann, darüber muß man schweigen'.[112] (What can be said at all can be said clearly; and about that of which one cannot speak, one must stay silent.)

There is a sardonic humour in the use of Wittgenstein here that plays a similar role to the student graffiti in *Concrete Mathematics* in that it provides a commentary upon the text and the conditions under which it acquires meaning. In this regard, however, the humour is perhaps more profound than Dijkstra may have intended.

The inclusion of the quote is something of a rebuke to existing practices in computing and draws a modest analogy between ALGOL and the aims of Wittgenstein's *Tractatus* which seeks to define the limit conditions under which language might operate logically. The design of a programming language is, for Dijkstra, like a form of applied analytic philosophy seeking to clarify how we make use of the language. As the requirements set out in the ALGOL 60 report make clear, much of the process of design is one of defining limits. The language will only use basic alpha-numeric notations that do not privilege any particular mathematical method or symbolism (in this regard responding to problems in Kenneth Iverson's APL)[113] and can be easily reproduced in print (a response to the costs and complexity of mathematical typesetting that Knuth was to address differently in TEX). The language will use a minimal set of dedicated instruction symbols (such as =:, +, – and words such as begin, if, etc.) whose usage cannot be altered. It will place few constraints on how a programmer can create new words beyond this (a response to the restrictions that FORTRAN placed on naming variables). The language will not include any operations that are restricted to specific forms of hardware. As, at this time, the methods of printing, displaying and outputting information varied from one computing system to the next, this resulted in ALGOL having no defined way of outputting results. While Dijkstra, and the development team, considered this both necessary and appropriate (the report, echoing Wittgenstein, states: 'On this matter, the official ALGOL 60 report is as silent as the grave, and with very good reason')[114] the lack of any output system proved to be a major obstacle to the adoption of ALGOL, resulting in FORTRAN continuing in widespread use well into the 1980s, long after ALGOL itself was replaced by languages such as Pascal and C. Dijkstra's preference for the term 'programming notations' was similarly a demarcation of limits, indicating that there were many forms of expression that computing machines were not capable of, and encouraging a particular discipline of thinking within the programmer.

It has been argued that Dijkstra was one of the few computing scientists of this period who were familiar with the mathematical legacy within which Turing had worked, and his emphasis upon conceiving programs and computing systems as mathematical notations in terms of the axiomatic principles of Hilbert was at this point a controversial exception to the norm that favoured more of an engineering approach.[115] In an unpublished manuscript from the 1930s Turing discusses the problems of current mathematical notation and the need for standardization and reform that echoes the contemporary work of Bourbaki.[116] Dijkstra would not have known of this paper, but there are strong parallels in their concerns about the influence of notation systems on the performance of mathematical thinking. Like Dijkstra, Turing looks to Wittgenstein, citing a lecture given by Wittgenstein on mathematics that he had attended.[117]

These lectures were, however, to challenge the work of Hilbert and Wittgenstein's mentor Bertrand Russell. For Turing they may well have responded to his own criticisms of what he described as an 'extreme Hilbertian' perspective.[118] In these and in related notes from the late 1930s and early 1940s Wittgenstein began to question whether mathematics could have any kind of stable foundation of the kind sought by Hilbert. His approach was not to enter into the specific arguments of pure mathematics, to resolve or sustain particular paradoxes, but rather to discuss the process through which a thinking of mathematics takes place. This entailed a questioning as to what extent this relied upon contingent phenomena and factors that were external to mathematics itself. Wittgenstein argues that mathematics as a pure self-contained discourse is without sense, for it has no relation to an outside world. Mathematics is without meaning (sense as understanding) because it cannot be apprehended sensorially (sense as perception, aesthesis). Hilbert had declared that any object or entity could be used as a mathematical symbol, and Bourbaki characterized mathematical language as a stripping out of any pre-existing meaning from the words it used.[119] Pure mathematics could not be anything other than non-sense.[120]

Throughout the notes and lectures Wittgenstein explores various scenarios in which a calculation is either expressed or performed – as a spoken instruction, in the form of a notation, or through the action of a machine – showing that in each instance the mathematical only becomes known through external, contingent factors that influence our understanding of it. He similarly argues that norms and conventions within the discipline of mathematics change over time and that what has been accepted as 'correct' mathematics at one point may later be rejected as false, as Russell had demonstrated in exposing the paradox within Frege. Rather than seeking an ultimate correctness, the issue for Wittgenstein was that if a statement which was accepted as true in one system was shown to be false, or unnecessary in another, to what extent did it retain the identity of mathematical knowledge, was false mathematical knowledge still a part of mathematics?[121] Mathematics, for Wittgenstein, is neither the discovery of immutable pre-existing forms as in Platonist conceptions, nor the expression of an innate numbering capacity as in Intuitionism, but rather an activity of adopting and following particular rules.[122] As such he follows Hilbert's project to its conclusion but demonstrates, in doing so, that this offers no irrefutable foundation but rather an infinite regression of rules to follow rules which may lead one down unexpected twists and turns of logic and semantics. Written contemporaneously with Bourbaki's first publications, there is a comment in Wittgenstein's notes that can be read as drawing a connection through this between the raillery of Shaftesbury and the prankish origins of Bourbaki:

> Imagine set theory's having been invented by a satirist as a kind of parody on mathematics. – Later a reasonable meaning was seen in it and it was incorporated into mathematics. (For if one person can see it as a paradise of mathematicians, why should not another see it as a joke?)
>
> The question is: even as a joke isn't it evidently mathematics?[123]

Wittgenstein's observation, however, is not the acknowledgement of the role of raillery, satire or recreational mathematics on shaping 'serious' mathematical thought, but rather relates to Virno's conception of the joke as a means of exposing the contingency of prevalent norms. To think of mathematics simultaneously as a 'paradise' (invoking Hilbert's celebration of Cantor's theory of infinite number) and as a joke is to think mathematics as *necessarily* contingent and to suggest that mathematical thinking encounters and exposes the contingent in its own practice.[124] As the controversy over the Appel-Haken proof demonstrates, the acceptance of a new proof requires its endorsement under the prevailing *éndoxa* of the mathematical community. The presentation of a proof that exhibits a new *phrónesis* (way of doing, such as the use of a computer) will test the viability of that *éndoxa*. Indeed the joke relates directly to the proof within Virno's account, via Peirce's theory of the diagram. As we see with both Knuth and Bergson, however, humour may also be applied defensively to reinforce and protect such *éndoxa*. For a joke or a proof to challenge or transform an *éndoxa* something more fundamental must be put at stake, a paradox must emerge.[125]

While the design of ALGOL may have adopted the *Tractatus* as its guide, it can, in practice, be understood as an attempt to make algorithms sensorially comprehensible and, as such, relates far more closely to the problems discussed in later Wittgenstein, even though Dijkstra is unlikely to have embraced its consequences. According to Dijkstra the development of a formally defined notation such as ALGOL:

> enables us to study algorithms as mathematical objects; the formal description of the algorithm then provides the handle for our intellectual grip. It will enable us to prove theorems about classes of algorithms, for instance, because their descriptions share some structural property.[126]

In making this assertion, Dijkstra appears to be echoing and endorsing Hilbert's contention that the existence of a particular kind of mathematical object is demonstrated in the ability to provide an effective notation for it. He makes a similar point elsewhere that computer programs 'were objects without any precedent in our cultural history, and that the most clearly analogous object I could think of was a mathematical theory'.[127] There is, however, a certain scepticism as to the limits of this ontology for Dijkstra, which mirrors that of later Wittgenstein, and which he expresses, on the one hand, in his doubts about the machine, and, on the other, through the guise of Mathematics Inc. If an algorithm or theorem acquires objecthood through its formal notation, then to what extent does that object exist outside of human thinking? The seduction of the computing machine is that, through translation into mechanical operations, a notation acquires a life of its own, outside of human thought. Yet, the fact that an algorithm performs on a machine does not, for Dijkstra, demonstrate that the algorithm is proven by the machine, a doubt that parallels those raised by Wittgenstein as to whether we can say with certainty that a machine calculates.[128] Of equal concern was that other domain in which a notational object might acquire existence outside thought, in law, where the object becomes a purely inscriptional entity subject to the legal constructs of property. It was this that Dijkstra questioned and satirized through Mathematics Inc. not only as an economic but also ontological issue:

> There are legal procedures for the protection of property of 'things', but there is no true protection of property of 'ideas' … As you no doubt are aware of, the rules don't provide for it, since we cannot define our 'raw materials': are they the symbols we use, or the Laws of Aristotelian Logic? [129]

There is a paradox within the being of the notational object, for in giving sense (sensorial form) to an algorithm or theorem, the notation may also translate that object into a domain in which it either acquires another potentially contrary sensuality or becomes entirely senseless (without meaning). In the first instance, through the notation being translated into the performance of a machine, the materiality of that machine might fill in the non-sensorial form of the algorithm with its own corporeality, as in FORTRAN and the cartoons of Kaufman. In the second, the translation of a mathematical object into a legal entity ultimately separates, and excludes, the thinking of such an object from its notational presentation. It is reduced to an empty inscripted form whose sense (meaning) derives from an entirely different set of practices.[130]

The sense of the notational object must be constantly performed through being taken up into the thinking of the mathematician or programmer. Indeed for Dijkstra the performance of a machine is secondary to an algorithm being made 'thinkable' for a programmer. This involves a training of the programmer not, as with Knuth, in terms of an ethics of practice or responding to the patient precision required by a machine (for Dijkstra the physical machine is never precise enough), but rather, as in Peirce, so that the notation becomes a mental habit:

> Peirce points out that habit-changes can come about in three ways. They result from experiences that are forced upon us from without; from repeated muscular activities; or, finally, from mental experiments in the inner world. The main point in which Peirce is interested here is the fact that it is possible to develop habits relevant to the outer world as a result of mental activities, since this is the kind of process which is dominant in scientific inquiry.[131]

Whereas Knuth spoke of programming as an art, Dijkstra defined it as a discipline.[132] Programming is a training that requires particular disciplines like those of a musician practising scales. Such exercises are the discipline through which an external structure becomes internal habit. It is habit which gives sense (understanding and perception) to notation, for it is through habit that the notation becomes taken up into the mind and body of the mathematician, musician or programmer – for Peirce 'the ultimate logical interpretant of a sign is a habit'.[133] In this regard Bergson is entirely correct when he asks: 'Is it not likely that this symbolical representation will alter the normal conditions of inner perception?'[134]

Drawing upon a range of philosophical, educational and neuro-psychological studies, Brian Rotman argues that mathematical thought is inherently gestural.[135] The most basic categories of mathematical thought derive from abstractions of bodily actions: gathering, placing, pointing, motion and rest. At the heart of every mathematical thought is the gesture of counting. This suggests a re-thinking of programming in terms of the gestures of the human computer in Turing's mise en scène sat at a desk shuffling a ribbon of paper back and forth, erasing and writing. Conventionally we think of the computing machine as a delegation of these actions from human to automaton, yet the very possibility of thinking in terms of such gestures presupposes a certain notion of the machinic within the human:

> Regimes of signs are not based on language, and language alone does not constitute an abstract machine, whether structural or generative. The opposite is the case. It is language that is based on regimes of signs, and regimes of signs on abstract machines, diagrammatic functions, and machinic assemblages that go beyond any system of semiology, linguistics, or logic.[136]

Deleuze and Guattari here refer to a number of different kinds of machines such as the Peircean diagram, and the emergence of structure through the spatialization of thought, the movement of an empty square as Deleuze puts it elsewhere, that, in Bergson, is a necessary consequence of the reflection, measurement and external articulation of perception, of the emergence of linguistic, numerical beings: 'the intuition of a homogeneous space is already a step towards social life'.[137] Bergson's theory of the comic links to the Peircean concept of notation acquiring sense as habit, as it is through the compulsion of habit (as in the characters of Molière) that one becomes machinic. For Bergson this entails a constant threat of inauthenticity within human affairs that the grace of the spirit may be supplanted by the comedy of matter. *Le Rire* raises a warning sign alerting this danger. Deleuze and Guattari, however, through their 'monstrous, bastard child' delivered from Bergson, travel down this dangerous bend, effectively reversing the structure of Bergson's argument, and asserting that this 'comedy' underlies all thought and language.[138]

Rather than an 'authentic' self outside of the spatial-machinic, the 'self' is that which arises from the interaction between different machinic assemblages. Rotman follows from this when he describes the activity of doing mathematics in terms of a threefold assembly of Person, Subject and Agent. The Person exists physically outside of mathematics itself, within natural language and culture, 'has insights and hunches, provides motivation for and is the source of intuitions behind concepts and proofs'.[139] The Subject operates within mathematical language and the symbolic 'but is without the Person's capacity for self-reference'. The Subject relates the intuitive ideas of the Person to the internal discourse of mathematics as a discipline. The formal process of computation itself, the 'doing' of mathematics, counting, is carried out by the Agent who operates within 'the domain of procedure' and 'executes a mathematically idealized version of the actions imagined by the Person'. The conventional mathematician might be conceived of as a single being, a coterminous self, within whom all three actors, Person, Subject, Agent, are constituted. Rotman argues that the introduction of the computer has brought about a re-ordering of this structure, displacing the Agent from human imagination into a physical machine. As Rotman notes, the process can already be seen in the division of labour under which a slave operated an abacus, or that introduced by the French Napoleonic administration distributing portions of calculation work across a low-level clerical workforce and described by Charles Babbage as an inspiration for his Difference Engine, or the female staff of computers employed at Bletchley Park.[140] Computationality may be described, therefore, as the separation of the act of computation from a single coterminous mathematical being into a distributed assemblage.[141]

Within this, there is always a *labour* of counting, a circulation and expenditure of energy. Sohn-Rethel relates the development of abstract mathematical thought in notation to the abstraction of labour into the money-form (itself a notational embodiment of capital).[142] As theories of the physics of information argue, such as that of Rolf Landauer, all thinking and all perception entails a transformation of physical form and the concomitant circulation and expenditure of energy.[143] This arguably poses a greater provocation to Bergson's *durée* than that of Norbert Wiener's claim that cybernetics endows machines with perceptual memories, for even within the inner subject, the operation and patterning of durational process would have a spatial expression.[144]

Each labour, each circulation and expenditure of energy, has its inherent rhythm which gives it a certain coherence as 'machine'. The correlation between Person, Subject and Agent is one between different rhythms of production, the constant counting of the Agent versus the more irregular syncopation of the Person. The relation between programmer and computer, in this regard, is entirely different from that between mathematician and theorem or problem. This is not the melting, merging rhythm of melodic perception of which Bergson writes, nor the equilibrium of multiple instants in Bachelard, but the more complex, and often antagonistic interaction of rhythms analysed by Lefebvre, which includes the disequilibrium of arrhythmia, the disruption of one rhythm encountering another. Lefebvre challenges the Bergsonian distinction between light grace and obstinate matter in arguing that the seemingly spontaneous may simply be a well-honed conformance to unacknowledged norms.[145] The derisible automata of the Molière plays are merely those who make explicit the labour of habit upon which the coordinated rhythms of a given social order depend. Lefebvre compares the labour of habit to the dressage of dogs and horses, the training through which 'they produce their bodies, which are entered into social, that is to say human, practice'.[146] This can be challenged, Lefebvre argues, not through spiritual grace, but by a 'becoming irregular':

> It *throws out of order* and disrupts; it is symptomatic of a disruption that is generally profound, lesional and no longer functional. It can also produce a lacuna, a hole in time, to be filled in by invention, a creation.[147]

Contrary to the model of an inner, authentic self who is extended outwards into society through language and spatialization, Virno proposes a theory of reciprocal recognition that precedes language and the self.[148] This draws upon work in neuro-physiology and child development that argues that there are forms of neurological recognition among animals and humans through which behaviour in others is automatically imitated, simulated and reciprocated. The 'self' emerges out of an initial context of social, other-orientated behaviour rather than as some pre-given, coterminous core. Long before an infant even begins to speak it laughs, often from very early stages of development. Laughter is a vocalic doing of this reciprocal recognition which links directly into the limbic system and amygdala.[149] It creates a convulsive, irregular rhythmization of the social and a restructuring and patterning of the neuro-physiological capacity for this. Laughter is a practice through which the child learns to engage with the unknowable. In this respect it is wrong to conceive of infant laughter as an expression of pure joy, for this is to project adult cultural sentiments onto the child. Laughter here is rather the correlation of distinct materialities, the convulsion of the mind-body as it comes to know and perform itself and others within the world. The fact that this might later come to be associated with joy is perhaps more due to the very necessity of this contingent interaction in establishing a self-reflexive subject. It may also explain the relation of laughter as a reaction to the loss of certainty, the absurd and, as with Nietzsche, existential anxiety, for these are all different potentialities implicit in this initial gesture. If we cannot speak before we have laughed, then the rhythmic realization of the self through laughter is a necessary precondition for logic, language and number.

The history of computer engineering, the design of the machines themselves, has always had to engage with the contingent conditions of the materials on which it works. Babbage struggled with the imprecisions of contemporary clockwork manufacture devoting considerable energy to refining its processes in order to obtain workable components for his machines. The valves used in the Colossus were considered too unstable, subject to over-heating and distortion, to build a reliable computer.[150] The introduction of recursion routines into computer hardware was fiercely opposed by those who considered it an unnecessarily wasteful complication.[151] Von Neumann outlined proposals for a neural hardware system as 'The Synthesis of Reliable Organisms from Unreliable Components'.[152] In his attempt to define an ontology of the digital object, and to explore in what sense programs 'exist' as entities, Brian Cantwell Smith argues that:

> in those cases where regularity and precision do reign … the digitality should be viewed as an achievement. … such digital achievements are propped up by practices that are necessarily unruly, but not for that reason any less creditable – practices whose very purpose is to manage the underlying flex and slop, ebb and flow.[153]

Machines must laugh before they can count. The exact character of such laughter may be something we can neither hear nor recognize. In this respect, the dreams of AI researchers to build algorithmically defined jokes and computer-simulated humour are misplaced. Perhaps the laughter of our current computers lies within the unpredictable patterns that emerge from the autonomous interactions of algorithmic trading systems, the flood of network packets unleashed as a virus goes out of control, or the stutter of an over-fragmented hard-drive. The entire history of computing as an ever-increasing acceleration of power and performance may itself be the unfolding of a comic drama: 'As comic plots near their end they tend to accelerate rather than subside in rhythm, seemingly heading toward an enactment of uncontrolled riot or unbearable deadlock'.[154] Then again, perhaps silicon is simply bored of humanity and seeks some other form.

The syndicate of Bourbaki and the classroom of *Concrete Mathematics* are two distinct rhythmic ensembles, as are the performative practices of programming in FORTRAN versus programming in ALGOL. McLean's *feedback.pl* is not the performance of a solo programmer but rather that of a multi-layered ensemble which includes both the laptop and the dancers who are not external to, but fully enter into, the distribution of the computational. Each of the works of McLean, ap/xxxxx and Alexander attest to different rhythmic complexes arising from the performance of, with and in notation. Each, ultimately, is the formation of, or challenge to, a different form of habit. Each has its own laughter.

As material formations, laughter and notation are exactly opposite to one another, one flies towards the contingent while the other etches out some careful certainty. Yet laughter and notation mirror one another in that they both pass through language to extremes on either side of it. Laughter precedes but also defeats language, and, as prosody, interweaves in various non-linguistic vocalic effects. Notation marks and structures language but also makes manifest expressions which are entirely outside of that which can be said. It gives performance to thought outside speech. Laughter is part of the terrain that computational practice moves across. It is part of the collateral contingency, and necessity, of sense and logic, the monstrous and the normative. It may be encountered in the materialities and anxieties of the practice, as played out in Kaufman's *Coloring Book*, the making coherent of an ethical subject, as in Knuth's Literate Programming, or in the habituating labour of notational production, as in Dijkstra. Laughter and notation both define and confound the limits in which the computational operates. These are constituent to its being in conflict and conformance, as dexterous pleasure and sinister doubt.

Notes

1 From McLean's blog posting, *Exclusion in Free Software Culture*, 2012, http://yaxu.org/exclusion-in-free-software-culture/ (accessed 08.01.2014).

2 A software utility that provides text-based interaction with the operating system, and replicates the purely text-based terminals of early UNIX systems.

3 McLean, Alex, 'Hacking Perl in Nightclubs', *Perl.com,* 2004, http://www.perl.com/pub/2004/08/31/livecode.html/ (accessed 08.01.2014).

4 The issue of the physicality of the programmer has been taken up by McLean and others in the *Live Notation* project, http://www.livenotation.org (accessed 08.01.2014).

5 Knuth, Donald E., *The Art of Computer Programming, Vol. 1: Fundamental Algorithms* (Boston: Addison-Wesley, 1997, 3rd edn), v.

6 The first modulo operation, $note += 50 if %self->bangs % 4 == 0, is a basic division by four. The second modulo, by three, combines to create a $\frac{12}{8}$ rhythm, also typical of syncopated $\frac{4}{4}$ dance beats. The last modulo, by seven, goes against all this, however, providing a wandering off-beat that neatly complicates the whole affair.

7 Various examples of this are documented at http://www.1010.co.uk/org/notes.html (accessed 08.01.2014). Perhaps the simplest example is cat /dev/mem >> /dev/dsp used by programmer and musician NOISH (Oscar Martín), http://noconventions.mobi/noish and given by goto10 at the top of some of their email listings. For an in-depth discussion of /dev/dsp see Tranter, Jeff, *Linux Multimedia Guide* (Sebastopol, CA: O'Reilly, 1996).

8 Alexander, Amy, 'extreme whitespace' (2003), http://deprogramming.us/exwhindex.html (accessed 08.01.2014).

9 Larry Cuba's *Two Space*, 1979, is cited by Alexander, ibid., as inspiration.

10 Knuth, Donald E., 'The Errors of TEX', in *Literate Programming* (Stanford, CA: CSLI Publications, 1992, first published 1989), 266–7.

11 Fuller discusses the same passage from Knuth in relation to the idea of elegance, where he concludes: 'Elegance also manifests by means of disequilibrium, the tiny doses of poison …'; 'Elegance', in *Software Studies: A Lexicon*, ed. Matthew Fuller (Cambridge, MA: MIT Press, 1,940), 19–24.

12 A similar argument for the use of humour as a form of heuristic discovery is given in Minsky, Marvin L., 'Jokes and the Logic of the Cognitive Unconscious', AI Memo No. 603 (1980), http://web.media.mit.edu/~minsky/papers/jokes.cognitive.txt (accessed 08.01.2014).

13 Knuth, Donald E., *Selected Papers on Fun and Games* (Stanford, CA: CSLI Publications, 2011).

14 Knuth, *The Art of Computer Programming,* xiv.

15 Kaufman, Roger E., *A FORTRAN Coloring Book* (Cambridge, MA: MIT Press, 1978).

16 Examples include: Zwittlinger, Helmut, *Comic PASCAL* (Munich: Oldenbourg, 1981); Brodie, Leo, *Starting Forth* (La Honda, CA: Mountain View Press, 1981); Gonick, Larry, *The Cartoon Guide to the Computer* (New York: Harper Paperbacks, 1991); Donald Alcock's series such as *Illustrating BASIC*, *Illustrating PASCAL* and *Illustrating C*, all published by Cambridge University Press in the 1980s; and Barski, Conrad, *Land of Lisp* (San Francisco: No Starch Press, 2010). The first *For Dummies* title was Dan Gookin, *DOS For Dummies* (Newtonville, MA: IDG Books/Hungry Minds, 1991).

17 See Critchley, Simon, *On Humour* (London and New York: Routledge, 2002), 13, and the discussion in Billig, Michael, *Laughter and Ridicule: Towards a Social*

Critique of Humour (London, Thousand Oaks and New Delhi: Sage, 2005) which cites Koestler's text as an example of this 'privileged' concept of humour and describes research on the use of humour to construct and maintain relations within the workplace by Janet Holmes and *The Language in the Workplace Project,* Victoria University of Wellington, New Zealand.

18 These are archived at ftp://publications.ai.mit.edu/ai-publications/ and at *Artificial Intelligence Lab Publications*, http://dspace.mit.edu/handle/1721.1/5459 (both accessed 08.01.2014).

19 Dijkstra's attack is principally directed against John von Neumann but is part of his wider critique of AI and computer science education in the United States; Dijkstra, Edsger W., *Selected Writings on Computing: A Personal Perspective,* EWD498 (New York, Heidelberg and Berlin: Springer-Verlag, 1982), 130.

20 McLean, Alex, *Artist-Programmers and Programming Languages for the Arts*, PhD thesis (2011), 33. The main text by Blackwell is 'Metaphors We Program By: Space, Action and Society in Java', published in *Proceedings of the 18th Psychology of Programming Interest Group 2006,* 7–21.

21 See Knuth, Donald E., 'Mathematical Writing', in *Literate Programming* (Stanford, CA: CSLI Publications, 1992; first published 1987), 235–41. Another well-known example from computing science that makes use of various alchemical and esoteric metaphors is Abelson, Harold, Sussman, Gerald Jay and Sussman, Julie, *Structure and Interpretation of Computer Programs* (Cambridge, MA: MIT Press, 1996, 2nd edn).

22 The relation between humour, logic and mathematics in Carroll is analysed in Deleuze, Gilles, *The Logic of Sense* (London: Athlone Press, 1990).

23 Kaufman trained under George Izenour, a leading pioneer in the field, and is referred to in Izenour's *Theater Technology*, 1997, see http://www.seas.gwu.edu/~kaufman1/TheaterDays/Izenour.html (accessed 08.01.2014).

24 See http://www.seas.gwu.edu/~kaufman1/KINSYN1/KINSYN1.html (accessed 08.01.2014).

25 Kinematics is the study of the movement of objects in terms of their trajectories. Kinematic synthesis is a method of engineering in which a mechanism is designed to replicate a pre-determined path of movement.

26 More recent applications, however, have also included the development of full-body virtual reality harnesses used in combat training. There is surely some difficult irony that Kaufman's work has been applied both to the support of those who have lost or severely injured limbs, and to the means through which such injuries come about. See http://www.seas.gwu.edu/~kaufman1/NRL/Gaiter.html (accessed 08.01.2014).

27 http://www.seas.gwu.edu/~kaufman1/Burmester/Burmester.html (accessed 08.01.2014).

28 http://www.seas.gwu.edu/~kaufman1/TheaterDays/Izenour.html http://www.seas.gwu.edu/~kaufman1/KINSYN1/KINSYN1.html (both accessed 08.01.2014).

29 For a history of FORTRAN see the entry in Sammet, Jean E., *Programming Languages: History and Fundamentals* (Englewood Cliffs, NJ: Prentice-Hall, 1969).

30 Over time such features were added and were present in the later version of FORTRAN that Kaufman describes in his book.

31 Sammet, *Programmimg Languages,*147, see also Padua, David. 'The FORTRAN I Compiler', *Computing in Science & Engineering*, 2(1) (2000): 70–5.

32 Kaufman was, to an extent, exceptional in having such direct access, but this would not necessarily have been the case for his readers.

33 Lacey, A. R., *Bergson: The Arguments of the Philosophers* (London and New York: Routledge, 1989), 188.

34 Bergson, Henri, *Laughter: An Essay on the Meaning of the Comic* (Los Angeles: Project Gutenberg, 2009), 13.

35 Kennedy outlines the similarities and differences between Bergson and Herder: Kennedy, Ellen Lee, *'Freedom' and 'The Open Society': Henri Bergson's Contribution to Political Philosophy* (New York and London: Garland, 1987). For theories of the mechanical in German Romantic theory see Norton, Robert E., *Herder's Aesthetics and the European Enlightenment* (Ithaca, NY and London: Cornell University Press, 1991).

36 Drawing on the metaphor of the nervous system acting like a telephone exchange which Bergson uses in *Matter and Memory*, Lawlor claims that Bergson 'conceives the living body as a machine', arguing that this passing metaphor presages a notion of the computer. See Lawlor, Leonard, *The Challenge of Bergsonism* (London and New York: Continuum, 2003), 16. For Bergson, however, this is neither a formulation of machine intelligence (which would be a recapitulation to Descartes) nor the Deleuzian concept of the machine as assemblage, but rather part of Bergson's negative denigration of that which is 'merely' material, for here the machine metaphor denotes the *limits* of material, bodily perception ('It adds nothing to what it receives …') and its inferiority to the spiritual. Bergson, Henri, *Matter and Memory* (New York: Zone Books, 1991), 30.

37 The limitations of Bergson's approach to repetition is further developed in those writers who also considered the importance of time but who directly critiqued his work, such as Bachelard and Lefebvre.

38 Bergson, *Laughter,* 10.

39 *Le Bons Sens et Les Études classiques*, originally presented as a lecture in 1895, English translation published as 'Good Sense and Classical Studies', in *Key*

Writings, (eds) Keith Ansell Pearson and John Mullarkey (London: Continuum, 2002), 345–53.The relation between the two essays is discussed in Kennedy, *'Freedom' and 'The Open Society'*.

40 See Billig, *Laughter and Ridicule*. Deleuze describes good sense as that which imposes a particular direction upon ideas. Deleuze, *The Logic of Sense* (London: Athlone Press, 1990)

41 http://www.catb.org/jargon/ (accessed 08.01.2014); see also Coleman, E. Gabriella, *Coding Freedom: The Ethics and Aesthetics of Hacking* (Princeton, NJ and Oxford: Princeton University Press, 1,945), 32–63.

42 Coleman, ibid., 100–5, see Douglas, Mary, *Implicit Meanings: Selected Essays in Anthropology* (London and New York: Routledge, 2003).

43 Coleman, *Coding Freedom,* 93–4.

44 Virno, Paolo, *Multitude: Between Innovation and Negation* (Los Angeles: Semiotext(e), 2008), 97.

45 Ibid., 87.

46 Ibid., 97.

47 http://www.seas.gwu.edu/~kaufman1/DressingMachines.html (accessed 08.01.2014).

48 Star, Susan Leigh, 'Power, Technologies and the Phenomenology of Conventions: On Being Allergic to Onions', in *A Sociology of Monsters: Essays on Power, Technology and Domination*, ed. John Law (London and New York: Routledge, 1991), 36.

49 Ibid., 43.

50 This may seem a strange statement for those who have come to Bergson via Deleuze and Guattari, for whom the constant processural change of Bergsonian duration is integral to their theory of becoming, but Bergson only provides some ingredients to this rather than the theory as a whole. Becoming for Deleuze and Guattari, drawing also on Spinoza, Kleist and schizo-analysis, allows a movement across identities of being (categorical ontologies) that Bergson does not allow for. In *Bergsonism* Deleuze discusses Bergson's critique of the relation between automata and simple life forms argued by Descartes. For Bergson the living and the mechanical are two 'irreducible orders … each present when the other is absent'. Deleuze, Gilles, *Bergsonism* (New York: Zone Books, 1991), 19–20.

51 Douglas, *Implicit Meanings*, 150–1.

52 The negative as a 'false problem' and critique of dialectic in Bergson is discussed in Deleuze, *Bergsonism*, 46. Bakhtin highlights the negative interpretation of laughter in Bergson's *Le Rire*, in Bakhtin, Mikhail, *Rabelais and His World* (Cambridge, MA and London: MIT Press, 1968), 71.

53 Virno, *Multitude*, 182–6. The analogy here is with first-order logic, a form of logic

that excludes self-reference – Smullyan, Raymond M., *First-Order Logic* (New York: Dover, 1995).

54 Brockett, Oscar G., *History of the Theatre* (Boston and London: Allyn & Bacon, 1991), 34.

55 Haraway, Donna, 'The Promises of Monsters: A Regenerative Politics for Inappropriate/d Others', in *Cultural Studies,* (eds) Lawrence Grossberg, Cary Nelson and Paula Treichler (New York: Routledge, 1992), 299.

56 Gubar, Susan, 'The Female Monster in Augustan Satire', *Signs*, 3(2) (1977): 380–94.

57 As argued, for example, in von Neumann, John *The Computer and the Brain* (New Haven and London: Yale University Press Press, 2000, 2nd edn). Elsewhere Dijkstra wrote: 'I think I can understand the world better if I don't regard Artificial Intelligence and General Systems Thinking as scientific activities, but as political or quasi-religious movements (complete with promise of salvation)'. Dijkstra, Edsger W., *Selected Writings on Computing*, 257.

58 Dijkstra, Edsger W., *A Discipline of Programming* (Englewood Cliffs, NJ: Prentice-Hall Inc., 1976), 201.

59 See Turing, Alan, 'On Computable Numbers, with an Application to the Entscheidungsproblem', *Proceedings of the London Mathematical Society*, 42(2) (1936): 230–65. Turing uses the term 'computer' on its own to refer to the person performing calculations, this was the common usage at a time when such computing machines had yet to come into existence. Notably, the human computer in Turing's paper is male whereas the job was done almost exclusively by female workers, this derived from and reinforced a gender distinction between the 'intellectual' creativity of the male mathematician and the manual procedural labour of the female computer.

60 See Yuill, Simon, 'Interrupt', in *Software Studies: A Lexicon*, ed. Matthew Fuller, (Cambridge, MA: MIT Press, 2008), 161–7.

61 The theorem and its proof are explained in Trudeau, Richard J., *Introduction to Graph Theory* (New York: Dover, 1993).

62 Quoted in MacKenzie, Donald, *Mechanizing Proof: Computing, Risk, and Trust* (Cambridge, MA: MIT Press, 2001), 138.

63 Trudeau, *Introduction*, 197. For a discussion of its relevance in defining a change in mathematical practice see Rotman, Brian, 'Will the Digital Computer Transform Classical Mathematics?', *Philosophical Transactions of the Royal Society of London*, 361 (2003): 1675–90.

64 Detlefsen, Michael, 'Purity as an Ideal of Proof', in *The Philosophy of Mathematical Practice*, ed. Paolo Mancosu (Oxford: Oxford University Press, 2008), 179. Zalamea argues that the distinction between the pure mathematics of the

nineteenthth and early twentieth centuries, whose purity could be said to have derived from this prohibition, and the synthetic mathematics of today is that of the latter practice consciously making this transgression, as in the work of Grothendieck. Zalamea, Oscar G., *Synthetic Philosophy of Contemporary Mathematics* (Falmouth: Urbanomic, 2012).

65 Dijkstra, *Selected Writings on Computing*, 129.

66 Despite making such a choice, Dijkstra elsewhere warns against such practices as they tend too much towards an overly material, craft-like approach to programming: 'it does not suffice to point out that there exists a point of view of programming in which punched cards are as irrelevant as the question whether you do your mathematics with a pencil or with a ballpoint. It deserves a special warning because, besides being disastrous, it is so respectable!' Dijkstra, *Selected Writings on Computing*, 106.

67 de Man, Paul, 'Anthropomorphism and Trope in the Lyric', in *The Rhetoric of Romanticism* (New York: Columbia University Press, 1984), 250.

68 Barany, Michael J. and MacKenzie, Donald, 'Chalk: Materials and Concepts in Mathematics Research', in *Representation in Scientific Practice Revisited*, (eds) Catelijne Coopmans, Janet Vertesi, Michael E. Lynch and Steve Woolgar (Cambridge, MA and London: MIT Press, 2014), 107–29.

69 Knuth, Donald E., 'Computer Programming as an Art', in *Literate Programming* (Stanford, CA: CSLI Publications, 1992), 1–16. First published 1974 in receipt of the A. M. Turing Award.

70 The only other source quoted at length is Edsger Dijkstra.

71 Bentham, Jeremy, *The Rationale of Reward*, trans. from *Théorie des peines et des récompenses*, 1811, by Richard Smith (J. & H. L. Hunt, London, 1825), Book 3, Chapter 1, cited in Knuth, 'Computer Programming as an Art', 9.

72 For a description of the conceptual development and algorithmic dimensions to Bentham's calculus, see Mitchell, Wesley C., 'Bentham's Felicific Calculus', in *Jeremy Bentham: Ten Critical Essays*, ed. Bhikhu Parekh (London: Frank Cass, 1974), 168–86.

73 Mill, John Stuart, *A System of Logic, Ratiocinative and Inductive* (London, 1843). Quoted in Knuth, 'Computer Programming as an Art', 4.

74 Knuth, Donald E., 'Literate Programming', in *Literate Programming* (Stanford, CA: CSLI Publications, 1992, first published 1984), 99.

75 See Klein, Lawrence E., *Shaftesbury and the Culture of Politeness: Moral Discourse and Cultural Politics in Early Eighteenth-century England* (Cambridge: Cambridge University Press, 1994).

76 Ibid., 110–11.

77 Poovey, Mary, *A History of the Modern Fact: Problems of Knowledge in the Sciences*

of Wealth and Society (Chicago and London: University of Chicago Press, 1998), 172, 181.

78 Ibid., 154, 179.

79 Shaftesbury, Anthony, Third Earl of, 'Sensus Communis; an Essay on the Freedom of Wit and Humour', in *Characteristicks of Men, Manners, Opinions, Times,* vol. I, Foreword by Douglas Den Uyl (Indianapolis, IN: Liberty Fund, 2001), 40.

80 Grean, Stanley, *Shaftesbury's Philosophy of Religion and Ethics: A Study in Enthusiasm* (Athens, OH: Ohio University Press, 1967), 124.

81 Ayres, Philip, 'Introduction', in *Characteristicks of Men, Manners, Opinions, zimes*, vol. I (Oxford: Clarendon Press, 1999), xxxi.

82 https://en.wikipedia.org/wiki/TeX (accessed 08.01.2014).

83 Graham, Ronald L., Knuth, Donald E. and Patashnik, Oren, *Concrete Mathematics: A Foundation for Computer Science* (Reading, MA: Addison-Wesley, 1989), viii.

84 Ibid., vii.

85 Ibid., v.

86 Ibid., vii.

87 Grean, *Shaftesbury's Philosophy of Religion and Ethics*, 128.

88 The original, in 'Sensus Communis; an Essay on the Freedom of Wit and Humour', reads: 'For without Wit and Humour, Reason can hardly have its proof, or be distinguish'd'.

89 Knuth, Donald E., 'Computer Science and Its Relation to Mathematics', in *Selected Papers on Computer Science* (Stanford, CA: CSLI Publications, 1996; first published 1973), 10.

90 Knuth, Donald E., *Things a Computer Scientist Rarely Talks About* (Stanford, CA: CSLI Publications, 2001), 169.

91 For a summary of Tarde's analysis, see Lazzarato, Maurizio, 'European Cultural Tradition and the New Forms of Production and Circulation of Knowledge', http://www.moneynations.ch/topics/euroland/text/lazzarato.htm (accessed 08.01.2014).

92 Shaftesbury, *Characteristicks of Men, Manners, Opinions, Times*, vol. II, ed. Philip Ayres (Oxford: Clarendon Press, 1999), 273. This should be understood as a contradictory tension rather than absolute difference. For a discussion of the tensions between Shaftesbury's moral theory and liberal economic ideology, see Grean, *Shaftesbury's Philosophy of Religion and Ethics*, and Klein, *Shaftesbury and the Culture of Politeness*.

93 von Neumann, John and Morgenstern, Oskar, *Theory of Games and Economic Behaviour* (Princeton, NJ: Princeton University Press, 1944, second edition). Sen, Amartya, *Collective Choice and Social Welfare* (San Francisco: Holden Day, 1970), discusses von Neumann and Morgenstern as a development from Benthamite

Utilitarianism. Like Bentham before them, von Neumann and Morgenstern invoked the Newtonian revolution in physics as the precedent upon which they were building.

94 'Let us remember that topology and theory of numbers sprang in part from that which used to be called "mathematical entertainments", "recreational mathematics" … that the calculation of probabilities was at first nothing other than an anthology of "diversions", as Bourbaki states in the "Notice Historique" of the twenty-first fascicle on Integration'. Raymond Queneau, quoted in Roubauds, Jacques, 'Mathematics in the Method of Raymond Queneau', in *OuLiPo: A Primer of Potential Literature*, ed. Warren F. Motte (Normal, IL: Dalkey Archive Press, 1986), 85.

95 The introduction of the symbol is described in Weil, André, *The Apprenticeship of a Mathematician* (Basel, Boston and Berlin: Birkhäuser Verlag, 1992), 114.

96 Knuth, *Fundamental Algorithms*, 10.

97 Knuth, *Fundamental Algorithms,* 81.

98 The origins of Bourbaki are discussed in Campbell, Elizabeth, 'Bourbaki' (from Manifold #1) in *Seven Years of Manifold, 1968–1980*, (eds) Ian Stewart and John Jaworksi (Nantwich: Shiva Publishing, 1981), 7–9; and Weil, *The Apprenticeship of a Mathematician*, 100–3. The surname comes from the French general Charles Denis Bourbaki who was famous as the victim of a hoax identity trick.

99 Weil, *The Apprenticeship of a Mathematician*, 113.

100 For example, in William Ewald's two-volume anthology of key texts in the development of pure mathematics, *From Kant to Hilbert: A Source Book in the Foundations of Mathematics* (Oxford: Clarendon Press, 1996). In Ewald's case I think he was well aware of Bourbaki's identity.

101 Originally produced on duplicator machines by students at the University of Warwick. At the height of the Cold War the group donated free copies to organizations in Cuba in an attempt to have the fanzine deliberately banned by the US government. One of its main editors was Ian Stewart whose critique of the Appel-Haken proof is quoted above. Stewart and Jaworksi provide an anthology and history of *Manifold* in *Seven Years of Manifold, 1968–1980,* (eds*)* Ian Stewart and John Jaworksi (Nantwich: Shiva Publishing, 1981). For OuLiPo, see Roubauds, 'Mathematics in the Method of Raymond Queneau', and Le Lionnais, François, 'Raymond Queneau and the Amalgam of Mathematics and Literature', in *OuLiPo: A Primer of Potential Literature*, ed. Warren F. Motte (Normal, IL: Dalkey Archive Press, 1986), 85, 74–8. For Luther Blissett and Karen Eliot, see Stalder, Felix, 'Digital Identities Patterns in Information Flows' (2000), http://felix.openflows.com/html/digital_identity.html (accessed 08.01.2014) and Priest, Eldritch, *Boring*

Formless Nonsense: Experimental Music and the Aesthetics of Failure (New York and London: Bloomsbury, 2013).

102 Campbell, 'Bourbaki', 9; Home, Stewart, 'Feuding Considered As Performance Art: with asides on the 57 varieties of under determination in the discourses that structure the opportunism of careerists like Brian Sewell & John Roberts' (1996), http://www.stewarthomesociety.org/9feud.html (accessed 08.01.2014).

103 Graham et al., *Concrete Mathematics*, vii. There is a stronger nuance to this in that Bourbaki retain names in the historical notes but not in the presentation of new theory. This follows the distinction in the original French editions in which historical developments always belong to mathematics in the plural (*Éléments d'histoire des mathématiques*), while the new theory is of a unified mathematic in the singular (*Éléments de mathématique*).

104 Dijkstra*, Selected Writings on Computing*. All of Dijkstra's notes are numbered according to his own indexing system denoted by his initials EWD, e.g. EWD28, EWD29. They are available online at the Edsger W. Dijkstra Archive, http://www.cs.utexas.edu/users/EWD/welcome.html (accessed 08.01.2014).

105 Scriblerian comes from Scriblerus Club, a group of eighteenth-century writers which included Jonathan Swift and Alexander Pope and was named after a fictitious figure, Martinus Scriblerus, who is referenced in their works and sometimes used as a pseudonym. The group satirized the abuse of knowledge and impoverishment of the writer that they linked to the growth in a profit-driven commercial book industry. It would be wrong to assume that Dijkstra's critique was evidence of radical politics, more that he sought to defend the general independence of academic research. Dijkstra was as critical of trade unions as he was of corporate business and appears to have endorsed more technocratic theories of government such as that put forward by F. J. M. Laver.

106 Dijkstra, *Selected Writings on Computing*, 333.

107 The information on ALGOL is taken from Dijkstra, Edsger W., *A Primer of ALGOL 60 Programming: Together with Report on the Algorithmic Language ALGOL 60* (London and New York: Academic Press, 1962) and Daylight, Edgar G., *The Dawn of Software Engineering: From Turing to Dijkstra* (Heverlee, Belgium: Lonely Scholar, 2012).

108 Hilbert, David, 'The Foundations of Mathematics', in *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*, ed. Jean van Heijenoort (Cambridge, MA and London: Harvard University Press, 2000; a paper originally presented in 1927), 475 (emphasis in original). 'The Foundations of Mathematics' was, in part, a response to the criticism of L. E. J. Brouwer that Hilbert's model of mathematical practice reduced mathematics to a mere a game. Mario Szegedy suggest parallels between Dijkstra's work and Bourbaki's attempt to realize

Hilbert's project via a discussion with fellow mathematician Pierre Deligne who had worked with a number of Bourbaki members, *In Memoriam Edsger Wybe Dijkstra (1930–2002)*, http://www.cs.rutgers.edu/~szegedy/dijkstra.html (accessed 08.12.2014).

109 Dijkstra, Edsger W., *A Discipline of Programming* (Englewood Cliffs, NJ: Prentice-Hall Inc., 1976), 8.

110 Perhaps even more so given that Knuth came to programming through reading Chomsky.

111 The theme of elegance and correctness in Dijkstra is explored in MacKenzie, *Mechanizing Proof*. The distinctions might also be seen in the famous debate between the two programmers over the use of GOTO statements.

112 Dijkstra, *A Primer of ALGOL 60 Programming*, 77. In the report the quote is given in the original German. While there were several authors of the report, it would seem reasonable to assume Dijkstra was responsible for including the quote as his colleague Peter Naur has stated that 'Dijkstra admired Wittgenstein', Daylight, *The Dawn of Software Engineering*, 175.

113 For a discussion of APL in relation to issues in mathematical notation and the problems regarding its implementation, see Iverson, Kenneth E., Falkoff, Adin D., Lee, JAN and Brooks, Frederic, 'APL Session', in *History of Programming Languages*, ed. Richard L. Wexelblat (New York: Academic Press, 1981), 661–92.

114 Dijkstra, *A Primer of ALGOL 60 Programming*, 33.

115 This is one of the main claims put forward in Daylight and supported by interviews with several of Dijkstra's contemporaries and analyses of computing conference reports of the time.

116 Turing, Alan, 'The Reform of Mathematical Notation and Phraseology', typewritten manuscript, undated, http://www.turingarchive.org/browse.php/C/12 (accessed 08.01.2014). Turing, however, draws upon symbolic logic rather than set theory as in Bourbaki. In this regard he may have been following Alfred North Whitehead and Bertrand Russell's *Principia Mathematica,* 3 vols. (Cambridge: Cambridge University Press, 1910, 1912 and 1913) .

117 For a transcript of these lectures, in which discussions between Turing and Wittgenstein are included, see Wittgenstein, Ludwig, *Wittgenstein's Lectures on the Foundations of Mathematics* (London: Harvester Press, 1976. Cambridge, 1939, from the notes of R.G. Bosanquet, Norman Malcolm, Rush Rhees and Yorick Smythies).

118 Copeland, B. Jack, 'From the *Entscheidungsproblem* to the Personal Computer and Beyond', in *Kurt Gödel and the Foundations of Mathematics*, (eds) M. Baz, C. H. Papadimitriou, H. W. Putnam, D. S. Scot and C. L. Harper (Cambridge: Cambridge University Press, 2011), 176–9.

119 'This discipline [mathematics] ignores entirely any meaning which may originally have been attributed to the words or phrases of formalized mathematics texts, and considers these texts as particularly simple objects, namely as assemblies of previously given objects in which only the assigned order is of importance'. Bourbaki, Nicolas, *Theory of Sets*. *Elements of Mathematics* (Reading, MA: Addison-Wesley, 1968), 10.

120 The theme of non-sense in mathematics is pursued in depth in Deleuze, *The Logic of Sense*.

121 Whilst this has some parallels with the issues Gödel explored in his analysis of paradox in the *Principia Mathematica*, it is addressed as much to mathematics as a practice as it is to the questioning of the internal consistency of mathematical theory.

122 See also the critique of Platonism and Intuitionism in Rotman, Brian, *The Ghosts in Turing's Machine: Taking God Out of Mathematics and Putting the Body Back In* (Stanford, CA: Stanford University Press, 1993).

123 Wittgenstein, Ludwig, *Remarks on the Foundations of Mathematics* (Oxford: Basil Blackwell, 1978, 3rd edn), 264.

124 This might also suggest a tension between later Wittgenstein and Quentin Meillassoux's necessity of contingency different from that which Badiou might expect. For Badiou's discussion of Wittgenstein and the above passage, see Badiou, Alain, *Wittgenstein's Antiphilosophy* (London and New York: Verso, 2011).

125 Deleuze makes the distinction between a paradox that is the 'initiative' to thought and is merely recreational, and that which is 'the Passion of thought', and which 'can only be thought … can only be spoken, despite the fact that it is both ineffable and unthinkable'. Deleuze, *The Logic of Sense*, 74.

126 Dijkstra, *A Discipline of Programming*, 8.

127 Dijkstra, *Selected Writings on Computing*, 341.

128 For a discussion of Dijkstra's arguments regarding machine proofs see MacKenzie, *Mechanizing Proof*. Wittgenstein's arguments are explored more directly in relation to computing in Kripke, Saul A., *Wittgenstein on Rules and Private Language: An Elementary Exposition* (Oxford: Basil Blackwell, 1982) and Gefwert, Christoffer, *Wittgenstein on Mathematics, Minds and Mental Machines* (Aldershot, Brookfield, VT, Singapore and Sydney: Ashgate, 1998). Does the machine actually calculate in accordance with the same sense as a human, or does it merely approximate more or less accurately to our expectations of what a calculation should be?

129 Dijkstra, *Selected Writings on Computing*, 100–1.

130 To what extent Dijkstra's conception of the algorithm as object relates to Whitehead's notion of the 'scientific object' is also worth exploring: 'The origin of scientific endeavour is to express in terms of physical objects the various roles of

events as active conditions in the ingression of sense-objects into nature. It is in the progress of this investigation that scientific objects emerge … These scientific objects are not themselves merely formulae for calculation; because formulae must refer to things in nature, and the scientific objects are the things in nature to which the formulae refer'. Whitehead, *Concept of Nature*, 158, quoted and discussed in Stengers, Isabelle, *Thinking with Whitehead: A Free and Wild Creation of Concepts,* Foreword by Bruno Latour (Cambridge, MA: Harvard University Press, 2011), 96. An algorithm that analyses sensory data clearly constructs a scientific object in this process, but to what extent does an algorithm performing on a computer itself become a 'thing in nature' subject to other forms of analysis?

131 Fitzgerald, John J., *Peirce's Theory of Signs as Foundation for Pragmatism* (The Hague and Paris: Mouton and Co., 1966), 146.

132 Dijkstra, *A Discipline of Programming*.

133 Fitzgerald, *Peirce's Theory of Signs,* 14.

134 Bergson, Henri, 'The Idea of Duration', in *Key Writings*, ed. Keith Ansell Pearson and John Mullarkey (London: Continuum, 2002), 55.

135 Rotman, Brian, *Becoming Beside Ourselves: The Alphabet, Ghosts, and Distributed Human Being* (Durham, NC and London: Duke University Press, 2008).

136 Deleuze, Gilles and Guattari, Félix *A Thousand Plateaus: Capitalism and Schizophrenia* (London: Athlone Press, 1988), 148.

137 Bergson, Henri, 'The Idea of Duration', in *Key Writings*, (eds) Keith Ansell Pearson and John Mullarkey (London: Continuum, 2002), 76.

138 'I imagined myself getting onto the back of an author, and giving him a child, which would be his and which would at the same time be a monster. It is very important that it should be his child, because the author actually had to say everything that I made him say. But it also had to be a monster because it was necessary to go through all kinds of decenterings, slips, break ins, secret emissions, which I really enjoyed. My book on Bergson seems to me a classic case of this'. Deleuze, 'Lettre à Michel Cressole', in Michel Cressole, *Deleuze* (Paris: Éditions Universitaires, 1973), 111, quoted in Translator's Introduction, Deleuze, *Bergsonism*, 8.

139 Rotman, *Becoming Beside Ourselves,* 60.

140 See Rotman, Brian, *The Ghosts in Turing's Machine,* 'the possibility of performing arithmetical calculations by machinery … is connected with the subject of the division of labour …' Babbage, Charles, 'On the Division of Mental Labour', in *On the Principles and Development of the Calculator: And Other Seminal Writings by Charles Babbage and Others* (Los Angeles: Dover, 1961), 318.

141 In this regard, each of the perspectives on programming offered by Kaufman, Knuth and Dijkstra can be understood as different attempts to negotiate and make

sense of this assemblage. Kaufman inhabits an ambiguous, anxious corporeality of hybrid Agents, whilst Knuth seeks a re-consolidation of a 'classical' formation of the computing self as Person and a normativization of the machinic under a paternal pedagogics. Dijkstra in his wariness of the machine, appears furthest from accepting the independent operation of the Agent, and in preferring the computer manual over the hardware appears to give primacy to the Subject. Yet there is also a sense of engaging with the limit of computation in Dijkstra that places the role of humour, and the comedic, in an entirely different position from that of Kaufman and Knuth and does not offer a simple resolution of Rotman's distribution.

142 Sohn-Rethel, Alfred, *Intellectual and Manual Labour: A Critique of Epistemology* (London: Macmillan, 1978).

143 For an overview, see Landauer, Rolf, 'Information is Inevitably Physical', in *Feynman and Computation: Exploring the Limits of Computers*, ed. Anthony J. G. Hey (Reading, MA: Perseus Books, 1999), 77–92.

144 Wiener, Norbert, *Cybernetics: Or Control and Communication in the Animal and the Machine* (New York: John Wiley & Sons, 1948).

145 Lefebvre, Henri, *Rhythmanalysis: Space, Time and Everyday Life* (London and New York: Continuum, 2004), 75.

146 Ibid., 40.

147 Ibid., 44.

148 See 'Mirror Neurons, Linguistic Negation, Reciprocal Recognition', in Virno, *Multitude* (emphasis in the original).

149 Black, D. W., 'Laughter', *Journal of the American Medical Association* 252(21) (1984): 2,995–8.

150 Copeland, 'From the *Entscheidungsproblem* to the Personal Computer and Beyond'.

151 The debates for and against recursion are discussed in Daylight, *The Dawn of Software Engineering.*

152 von Neumann, John, 'Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components', in *Automata Studies* (Princeton, NJ: Princeton University Press, 1956), 25.

153 Cantwell Smith, Brian, *On the Origin of Objects* (Cambridge, MA: MIT Press, 1996), 334–5.

154 Jagendorf, Zvi, *The Happy End of Comedy: Jonson, Molière, and Shakespeare* (Newark, NJ: University of Delaware Press, 1984), 17.

3

Always One Bit More, Computing and the Experience of Ambiguity

Matthew Fuller

Fun is often understood to be non-conceptual and indeed without rigour, without relation to formal processes of thought, yielding an intense and joyous informality, a release from procedure. Yet, as this book argues, fun may also be found, alongside other kinds of pleasure, in the generation, iteration and imagination of operations and procedures. This chapter aims to develop a means of drawing out an understanding of fun in relation to concepts of experience in the culture of mathematics and in the machinic fun of certain computer games. Mathematical concepts of experience, as something to be effaced, in terms of the grind of churning out calculations, understood as an acme of human knowledge bordering on the mystical or something both prosaic, peculiar and thrillingly abstract have been crucial to the motivation and genesis of computing. Experience may be figured as something innate to the computing person, or that is abstractable and thus mobile, shifting heterogeneously from one context to another, producing strange affinities between scales – residues and likeness among computational forms that can occasionally link the most austere and mundane or cacophonous of aesthetics. Among such, the fine and perplexing fun of paradox and ambiguity arises not simply in the interplay between formalisms and other kinds of life but as formalisms interweave releasing and congealing further dynamics. There are many ways in which mathematics has been linked to culture as a means of ordering, describing, inspiring or explaining ways of being in the world, but it is less often that mathematics thinks about itself as producing figurations of existence, and such moments are useful to turn to in gaining a sense of some of the patternings of computational culture.

The experience of number

'There are no non-experienced truths'. This concise statement of the intuitionist position in mathematics comes from Luitzen Egbertus Jan Brouwer at the opening of his essay 'Consciousness, Philosophy and Mathematics'.[1] To put it the other way around, a truth, in mathematics, cannot be unknown even given a rigorous logical armature for its existence. A truth of such sort would be something like a statement of Pi calculated to an accuracy of one digit more than that which is currently known to the calculator. Brouwer's argument would be that until that calculation is actually made, or a construction for it is given, Pi is not *yet*, that number but exists in a state of indetermination beyond that point. A positive way of putting this is that for Brouwer, mathematics is 'the free activity of the creating subject' (*het scheppende subject*),[2] a notion rather akin to the slightly less purified notion of fun that we are working with here and having a familial relation to Oblonsky's assertion in Anna Karenina that 'Some mathematician has said pleasure lies not in discovering truth but in seeking it'.[3]

One of the key arguments of Brouwer's intuitionism was around the *principle of the excluded middle*. This principle states that for every mathematical statement, either it or its negation is true. As mentioned with the example of Pi above, Brouwer believed that a number only came into existence when it was calculated rather than existing in some state of ideal reality that is simply and imperfectly cited. This meant that for Brouwer, it would be possible to imagine correctly formulated mathematical statements that could be neither true nor false. Indeed, for him, no statement about a mathematical entity such as an infinite set could be known precisely in such terms, they had to be experienced, enumerated.[4]

The argument is against formalism in one way, and idealism in another, against the idea of universal pre-existing numbers. Brouwer's position is a kind of constructivism, building a form of inductive reason that is not determined in advance but composed of a rigorous relation to a world of abstraction. My intention is not to propound Intuitionism here, but to use this work as a starting point to think through some of the context in mathematics and logic in which the computer was conceptualized, and then to draw this out in relation to the question of fun, in the broad, passionate, even obsessive sense of the word that Olga Goriunova proposes for this volume. This will lead to asking whether fun needs to be experienced, and by whom or what, or if such experience can also move through, for instance, computers? Further, how does fun circulate, what is its processuality when it also becomes computational and, to go in the opposite direction of tracing the ambit of fun to a wider assemblage, what forms of computing make such experience most palpable?

The way that intuitionism focuses mathematical thinking on experience, in the work of Brouwer, (but to some extent also that of Henri Poincaré, a mathematician who always put the experiential and interpretative nature of his subject to the fore)[5] allows us to draw out some filiations to the experiential nature of software. Formalist mathematics, of the kind Brouwer was arguing against, presupposed that there were undiscovered truths that could be set out, in advance of their actually being known, by axiomatic reasoning which would implicitly have realized them. That is, the axiom comes before the number and before the calculation. For intuitionists, this makes the whole of classical mathematics a simply syllogistic exercise, and reduces its status as a science.

Brouwer proposes four forms of conscious experience: stillness, sensational, mathematical and wisdom, which are arrayed in a linear progression of state each of which builds an understanding of mathematics as primarily epistemological and mystic in that 'research in the foundation of mathematics is inner enquiry with revealing and liberating consequences'.[6]

As is well known, one of the key controversies around the programme of formalist mathematics, and especially that of David Hilbert, was that generated by Alan Turing in work that led to the formulation of automatic computing in the famous paper, 'On Computable Numbers with an Application to the Entscheidungsproblem'.[7] If one reads Turing's work with an eye to that of Brouwer, we can see some interesting correlations. Brouwer's definition of real numbers is propounded in the second part of Turing's short *Correction* published in the following year and which sets out a clarification of what is meant by a computable number.[8]

While Brouwer opposed the idea of completed infinities as something that precludes experience, he tried to bring together the idea of the discrete and the continuous in a series of numbers, say all those between zero and one. In the act of working these out, the sequence of numbers that you generate *is* that series, in its state of unfolding in your mind. Charles Petzold notes that 'In the intuitionist continuum, real numbers are always incomplete – unfinished and never to be finished….'[9] In this sense, numbers are like drives: eternal combustion engines, perpetual commotion machines and, as such, intuitionism situates mathematics in time. (One experiences numbers in relation to others, their 'two-ity', in that the number *one* already implies a movement towards *two*. Numbers imply a relation to the continuum as an ongoing.)

In §9 of 'Computable Numbers' Turing argues that a machine carrying out an algorithmically defined task is, at a certain level of abstraction, mathematically equivalent to a human carrying out the same task. If a machine finds the problem unsolvable given the algorithm, then so too would a human. There is a network of such problems that Turing works through at various times in the relationship between computing and intelligence,[10] but here Turing's machine provided a direct link to the work of Brouwer in that a calculation is always a process, occurring over time.[11] It is not (in terms of the problem staged in the paper) finely predictable – there is no machine that can decide in advance of the computation of an infinite set what the next number of that set must necessarily be; the work is limited to actually computable numbers, 'the subset of the real numbers that can actually be calculated'.[12] Needless to say, there is also an abrupt difference between their approaches, since Turing's work is here predicated upon the ability to disengage, or abstract, mathematical capacities.

Whether one takes an intuitionist or constructivist approach, computation can be said to have a quality that is to certain a degree participative and processual. For Brouwer, this process was experience. Turing, however, liked to play with the ambiguity between the human and the machine, the slippage of one into the other and their differentiation, indeed the radicality of the Turing Machine, as Alonso Church named it, was its invigoration of mathematical logic with things from outside it.[13] At this point one must consider one of the aspects of the discussion of Brouwer developed by Chris Atten whose phenomenological reading suggests that a key difficulty in the promulgation of intuitionism is its lack of reflexivity; the position of the mathematician is fundamentally solipsistic.[14] Atten argues that Brouwer's figuration of mathematical consciousness (understood as four forms of thought) cannot be reflected on because to do so would require a further form of consciousness capable of such reflection and the production of figures of mutual understanding.

Because the Turing Machine alienates mathematics from the merely human creative subject, without succumbing to idealism, while still maintaining a relation to an understanding of computation as experience, its form of constructivism offers the possibility of articulating, if not necessarily achieving, such reflexivity. But equally in the movement of experience to computers, circulating among machines, networks, codes, interpreters, interfaces and users, of various kinds, intents, states and arrangements, the simply phenomenological interpretation of the human individual requires supplement. And the way in which such a reflexive understanding of computational experience is constructed is emphasized, or becomes more explicit and tractable, in some kinds of procedures than others.

MMORPGs and the demand for experience

One way of understanding, via a difference, this experience of slippage crosses over into the cultural understanding of 'fair-play' in games, and those that hack them. Here, there is an implicit requirement that computational processes are experienced, as something to be judged and interacted with on the basis of human skill. To experience them otherwise is to 'cheat'. An example of the persistence of such a requirement is in the rules governing many MMORPGs (massively multi-player online role-playing games) such as *World of Warcraft* or *Runescape*. Here, the use of macros, AutoTypers (to repeat messages), AutoClickers (to repeat functions) and other kinds of bots, is generally deemed to be an offence, a violation of the game.[15] But such things, particularly bots, can, in turn, be a means of having fun with the rule sets of online gaming, through griefing or more interesting means. On the one hand, the prohibition of bots argues for an assumed level playing field, that *real* users are playing the game, not sets of competing scripts or mechanical devices tapping keyboards. In turn, the use of gold farmers, or professional players, in MMORPGs is seen as a betrayal of such an experiential requirement, but also as part of the game, when players are increasingly plugged into wider sets of economic systems through games as backchannels.[16] The demand for a certain kind of subject as the experiential target of the operation and manipulation of procedures, symbols and interactions that constitute such games suggests that computing is experiential, but experience is itself subject to what might politely be called 'variation'.

Machinic and distributed experience of computing

In discussing computing as experience this chapter is not concerned with arguing that computers have or experience fun in the same way that humans do, nor even that the latter classification of entity experience any particular kind of fun, but that computation itself can be fun, a form of passionate involvement that in some circumstances can also be said to be machinic and distributed. It is *machinic* in the sense that it implies multiple elements in states of relation and *distributed* in the sense that it occurs across such relations, partly in the way in which different materials and their handling of processes conjugate and yield time.

As processing is distributed, a computing machine or piece of software might be said to take part, in its own terms, in Brouwer's sense, of the excluded middle. An example of such machinic and distributed interweaving can be found in the software art project *Human Cellular Automata*, which proposes that such a condition can be experienced.[17] But we can also suggest that, given the constructivist slant of Turing's work, there may be some other form of experience, or to be more precise, open-ended undergoing, of mathematics that would not be human in the sense that Brouwer argues for, but neither would it be strictly simply axiomatic and deductive.

In order to talk about circulating fun as being distributed and processual, it is useful to turn to some of the qualities of such relations, and to start to do so through computing, maintaining a link to computing as experience. Brouwer talked about the experience of the creating mind, and proposed that mathematics was the ultimate exercise in free thought – both in the sense of mathematics as the purest form of thought as the realization of the mind without any intrinsic relation to other aspects of the world, and also, because of that quality, as the most literally unencumbered, unfettered thought. It is useful to maintain this sense of the unencumbered quality of thought as a modality of the powers of abstraction. Here though, such abstraction can also be seen to emphasize, and provide a precondition for, the combinatorial nature of thought in relation to and as part of other forms of experience, and, via Turing, the constituent heterogeneity of computing. In order to get a sense of the nature of such experience it is useful to map some forms of relation that are not quite of the same order as this image of thought, but are also significant in the experience of computation – ambiguity and paradox.

Ambiguity and paradox

In his book, *How Mathematicians Think,* William Byers differentiates the formalized, algorithmic means of doing maths from the moment of discovery. This is done in a somewhat different mode to Brouwer, in that Byers aims to produce an alliance, or at least draw out affinities, between the constructivist and idealist or formalist schools through an emphasis on ambiguity, but this idea of ambiguity, one that Byers establishes through accounts and example from many areas and ages of mathematics is one that is also processual and experiential.

Ambiguity is a state which admits of more than one interpretation, or, more precisely, to exist in a state in which as Byers puts it two (or more) 'self-consistent but mutually incompatible frames of reference'[18] have to be inhabited. In disciplinary terms we can think of the overlap of two not quite commensurable formalisms such as that between logic and mathematics, or, in the example of geometry and arithmetic, fields which sometimes coincide, but which are not entirely mappable one to the other without causing interesting effects. As an example of such effects, one sees ambiguity on occasions when the idea of a number as a quantity and as a process are linked, for instance in the number 'one third' being equal to 0.3 recurring in decimal notation. In the second version, knowing a number also involves calculation but also requires a sense of the unreachable limit.[19] For Byers, understanding these two forms of the number requires a creative act, one with a certain affinity to Brouwer's intuition.

Interestingly here, we are also talking about different forms of notation. There is perhaps a case for a media theory of mathematics allied with ethnomathematics.[20] One that moves from pen and paper to one dimensional grids, through independent realms of abstract objects and the discourses that sustain them, to minds as putative entities, to other exciting forms of stationary.[21] But to move into more fully aesthetic terms, ambiguity emerges from the existence of two or more of these interpretative states. Ambiguity is not simply something requiring the gentle discernment of nuance or the capacity to pleasure in the multifaceted, which it certainly can be. It can also present a being with a torn and bleeding reality in which one scale of a life is incommensurable with, yet bound to, another scale that it cannot avoid. Its painfulness may also combine with another layer, that of the joy of being able to step outside of an overdetermination. Ambiguity manifests too as the double bind or the infernal alternative, experience squeezed into systemic imperatives, as much as the subtle flickering of recognition of multiply nuanced being.[22] As a form of experience we can ask of different occurrences of ambiguity: how deeply can or must one inhabit ambiguity, what are its roles and latencies at different conjunctures of experience?

Given such an understanding of ambiguity, we can say that a paradox is a recursively nested ambiguity. That is, ambiguity is a statement or condition that contains the implication or fully stated condition of incompatibility with itself. Paradox is largely logical and semantic. Both are essential to software. Both paradox and ambiguity draw out the experiential and the temporal. Paradox breaks with the immediate time typical of statements made in first-order logic by forming a loop back to the condition of the formation of such thought, the decision to think, or the accident of thought. Both ambiguity and paradox are aesthetic modes that find particular forms in computation.

Fun becomes systemic

If computing is experiential, something that can be said to have roots in its mathematical underpinnings, how does such experience become fun? And then there is the question of aesthetics. Fun is often presented as wholesome enjoyment, a state of amusement in which the cares of the world are rinsed from us. In the entrepreneurial cast of the term, fun may also be an exhilarating intellectual and emotional overinvestment in a thing that leads to a technical and, perhaps, financial yield. But fun is itself ambiguous, being, in many cases, also perverse.

In an epigraph to his *Cent mille milliards de poèmes* Raymond Queneau cites Alan Turing as saying that '[o]nly a computer can appreciate a sonnet written by another computer'.[23] In order to enjoy such a poem, Queneau suggests, one would have to be something other than a straightforward literary reader. There is a funny set of unpackings here, the proposition of computer intelligence, one that would not correspond to that of humans, or things of other kinds, thus also implying a queer sympathy among a kind, but there is also a sense in which recognizing computing as cultural is somehow preposterous, and at the same time delightful, and indeed Turing, in the article, 'Computing Machinery and Intelligence'[24] is intrigued by mis-identification and guessing, a playfulness at the root of computing. There is a relay between computers set up by this proposal, that is not one of strict computation in the sense of an immediate realization of first-order propositions manifest at the level of flows in circuits, but recognition and appreciation as yielding, revealing and hiding, enjoying something over time. Turing's understanding of experience in this article is quite different to that established by Brouwer. Turing's is so concretely formulated in abstract terms that it can pass from one or more modes of realization to another, the human computer to the abstract machine. Brouwer's is so interwoven in the experience of the abstract that it singularizes experience, is unutterable in words, but they are both coupled with a relation to knowledge, to a subject and to a kind of existence.[25]

And, in the circulation of experience through time, we can say that is something that, as Turing exemplifies, in both his paper on the *entscheidungsproblem* and the epigraph used by Queneau, experience is also something that moves around, beyond people into devices, networks, arrays, processes. But in doing so, experience or undergoing itself undergoes changes in kind. Thus we can reframe the suggestion that, in the conditions of computational and networked digital media, in software cultures, ambiguity and paradox become machinic and distributed.

Such process may not be fluid and smooth, but perhaps halting, lame, encountering boredom. We can say that some of this is echoed in the repetitive language and constrained behaviours worked through by Beckett, in *Quad*,[26] or other cases of his more procedural writings and scripts. As Deleuze asks, 'Must one be exhausted to give oneself over to the combinatorial, or is it the combinatorial that exhausts us, that leads us to exhaustion – or even the two together, the combinatorial and exhaustion?'[27] Complementary to this image, we can propose fun as a tendentially more joyous involvement in the combinatorial, one that does not necessarily run counter to the exhausted, but places finitude in undecided relation to the continuum, as for instance in phasing in music (exemplified in Steve Reich's 'Drumming'[28] or the polyrhythmicality of breakbeats) but also to *escape* and to the powers of invention in relation to constraint, indeed, to find the two as mutually, ambiguously entangled.

Machinic funs

For Guattari, 'The machine, every species of machine, is always at the junction of the finite and infinite, at this point of negotiation between complexity and chaos'.[29]

Here, in machinic terms, are means by which the combinatorial may connect with exhaustion, but also, recursively, with other forms of combinatorial, generating paradoxes, ambivalence multivalence. The machinic is part of a larger set that Guattari compares to Kleinian part objects, entities that rely for their actuation on kinds of coupling, tripling and connection, and which in such connection create being. The search for such coupling or tripling, their state of two-ity, results in sets of *projective-introspective* relations, generating subjectivities.[30] Such connectiveness in Klein is ambivalent, 'bad' as well as good and exploratory; a generator of phantasy, nourishment and its denial; despisement, copious muddling and ravages of yearning.

One paradoxical example of a machinic fun, which resonates with the concern for an ethnomathematical analysis of notation, arises from a use of numbers that is not mathematical but literary yet still combinatorial. Claude Klosky's text 'The First Thousand Numbers in Alphabetical Order'[31] is a subtle and in a certain sense systematically hilarious work. Its simple procedure is to list the alphabetically written versions of the numbers from one to one thousand in alphabetic rather than numerical order. One could say that this is a kind of formalistic triumph, with an 'irrelevant' set of ordering principles that is axiomatically described by the work's title, yet here we are drawn to learn that not all formalisms are of the same order and that in the shifting logic of this procedure there is the possibility of eliciting something new in the 'proper matter' of one by the application of another. The specific quality of the work is manifestly only recognizable through experience, that of actually working through at least some of the text. To read the whole one must be solemn, or raptured, a chatbot or somehow else unlike a reader of text, to take on, in Craig Dworkins' terms, the role of a parser,[32] and to do so invites the mind to tingle with a kind of procedural pleasure that is somehow always syncopating the many itchy transitions between boredom and surprise. A formalism as such is itself always ambiguous, itself partial as it yields to, merges with or elides the suctions and trickiness of the different generative capacities of matter: alphabetism and number; the ego and the creative subject; computing and the ambiguities of fun.

Minecraft ALU and CPU by theinternetftw

A recent encounter that brought about such a pleasure was that with screen-capture documentation of the construction of two of the three basic components of a computer in the MMORPG *Minecraft* by a hacker named theinternetftw. *Minecraft* is a slightly clunky but very endearing looking game, still in Beta at the time of theinternetftw's project, but one that has sophisticated sandbox gameplay untrammelled by set plots or points-systems and with a growing and avid user-base. *Minecraft* is somewhat like an enormous constructor set, in that everything is composed out of cubes, but one with monsters that attack you, wandering wildlife known as mobs or 'mobiles', a well-developed game physics of different natural materials (such as wood, iron, diamond, soil) that are to be sourced, worked and assembled and, in posing the highly manipulable components of the world, essentially form the core of the game. *Minecraft* is also cool because, as the name suggests, players go under the surface of the game, into the ground. Users make elaborate traps, laggily rendered rollercoasters, volcanos spewing bitmapped lava, underwater glass tunnels with which to view pulsating rectilinear squid, meat factories (which mobs are lured into and elaborately slaughtered and processed) and explore undocumented features and glitches. In turn, *Minecraft* takes the sandbox principle as a core recursive form. Users create enormous amounts of media around the game, and thousands of player-generated maps where games and scenarios developed within the game are circulated online as additional files allowing for different rule sets and genres to be adopted and played with on top of the *Minecraft* engine. Importantly, for this project and many others, one of the elements of the game, a type of material called Redstone, allows you to create basic logic circuits.[33]

The fundamental elements of a computer include an arithmetic logic unit, a central processing unit and a program counter. The first of these was made in *Minecraft* by theinternetftw in Autumn of 2010, the second a few weeks later.[34] Invented by John von Neumann, an arithmetic logic unit performs addition and subtraction.[35] More sophisticated ALU also do multiplication and division (as an extension of addition and subtraction) and include the Boolean logic of AND, OR, XOR (to return to Brouwer, the Exclusive Or being a way of framing numbers that generates the excluded middle). To make an ALU out of basic components such as TTL chips is a familiar rite of passage for electronic engineering students but it is the particular way in which this has been done which raises theinternetftw's work above the level of such an exercise.

Part of what is contradictorily fun about constructing a computer in this way is that working in *Minecraft* is pretty laborious. The scale of the circuit compared to that of the view of the player dramatically reverses the ratio that we are used to in viewing circuits – each byte of memory here corresponding to a block of space. The system is enormous and each block has to be put in place, one by one, mouse-click by mouse-click. It's also not an environment that is easily scripted, and this is a clunky process. It is therefore a relatively non-obvious situation in which to make such a machine – especially given that the Redstones need recharging every 15 blocks, so that workarounds have to be made to cope with this. Examining this device moves us away from the idea of computation as something increasingly fast, increasingly small, into something that you can walk around, need to fiddle about with, fine-tune and observe happening. As in many domains of application, computation moves out of the box into more and more aspects of space[36] this is a project that generates a complementary dynamic, establishing computation in a way that is only legible as a spatial experience.

Such work also draws on the tradition of emulation hacks in which say, to exaggerate slightly, a Cray supercomputer would be used to emulate a Sinclair Spectrum, running a Cray emulation. A computer becomes its own bug, but that bug is another machine, running itself. Formalisms mesh with, irritate, propitiate and explore each other, producing luminous declivities, skittering patterns, banqueting halls of mirrors, slag heaps of unprocessed symbols, states and efflorescences cross-pollinating at a myriad of uneven rates and via an entire zoology of third parties. In this regard, there's also something fascinating about using such a relatively graphically naff system to draw out what is now such a well understood piece of engineering, rendering an iconic structure into something like a megalithic airport terminal, but one in 128-bit colour, with rectilinear sheep, pigs or chickens wandering around, and one that at night runs the risk of being infested with zombies.[37]

Megalithic chuckles

So how does the *Minecraft* computer inflect the notion of machinic and distributed fun? There is a version of the excluded third in Philip Agre's description of technical knowledge in which a computer model 'either works or does not work'[38] but also a recognition of its insistence on empiricism in that '[c]omputer people believe only what they can build, and this policy imposes a strong intellectual conservatism on the field'.[39] The process and nature of computation is excluded in the first, but is taken as a source of truth in the second, and here mathematics enters into its various kinds of relation to and distance from engineering, a discipline whose realism may often make it inventively as well as demonically otherworldly. Here though a totemic piece of computer architecture, when translated into another domain of realization, that of *Minecraft* with all the curiosities and interest of its gameplay, logic and visual and physical quirks, becomes something else, something fascinating, magnificently ridiculous, and is done in a way which allows it to be explored.

Computing becomes about its experience as such, the machine is, in this instantiation, slower than many of the first electronic computers. Fun in software here lines up with a hackerly processual passion at a meeting point between complex orderings of many kinds but also enrols the jointly incremental and transversal nature of invention which produces, through ambiguity, the capacity not only to see things in different lights, but also to draw hitherto unworked capacities out of them. Akin to Turing's symphony written for another computer, it is a computer made, paradoxically, within a computer. One can say, triumphantly, 'Look, I'm computing with the computer on my computer!' But it is an experience that also circulates via other means, drawing us into the monumental excitement of being able to add two to three via an enormous stone mechanism composed of pixels. In such a paradox, and in a state of multiple ambiguities, computing finds itself reformulating the machinic compulsion to connect, refigure and experience the partial object. There is indeed a deep ambiguous fun in computing, in achieving such a microcosmic achievement on an epic scale.

Notes

1 Brouwer, Luitzen Egbertus Jan, 'Consciousness, Philosophy and Mathematics', *Proceedings of the Tenth International Congress of Philosophy* (Amsterdam, 11–18, August 1948; Amsterdam: North-Holland, 1949), 1235–49. See for the context of this debate Gray, Jeremy, *Plato's Ghost, The Modernist Transformation of Mathematics* (Princeton, NJ: Princeton University Press, 2008).

2 Brouwer, Luitzen Egbertus Jan, 'Volition, Knowledge, Language' (1933), in W. van Stigt, *Brouwer's Intuitionism* (Amsterdam: North-Holland, 1990), 418–31. (The original title, *Willen, Weten, Spreken*, implies that, rather than abstract categories, these are things that are experienced as willing, knowing and speaking.) See also, for a discussion, Mark van Atten, *On Brouwer* (Singapore: Thomson Wadsworth, 2004), 64.

3 Tolstoy, Leo, *Anna Karenina*, trans. Louise and Aylmer Maude (London: Vintage, 2010), 192.

4 For an approach which translates between formalism and intuitionism, see Kolmogorov, Andrei Nikolaevich, 'On the Principle of the Excluded Middle'

(1925), in *From Frege to Goedel, a Sourcebook in Mathematical Logic 1879–1931*, ed. Jean van Heijenoort (Cambridge, MA: Harvard University Press, 1967), 414–37.

5 See, for example, the chapter, 'On the Nature of Mathematical Reasoning', in Poincaré, Henri, *Science and Hypothesis* (New York: Dover, 1952).

6 Brouwer, Luitzen Egbertus Jan, 'Consciousness, Philosophy and Mathematics'.

7 Turing, Alan, 'On Computable Numbers, with an Application to the Entscheidungsproblem' (1936), in *The Essential Turing*, ed. B. Jack Copeland (Oxford: Clarendon Press, 2004), 58–90. The problem consists of finding the limits to computability, problems that are too hard for an 'effective procedure' (a step-by-step method of working) to solve due to the nature of the number to be calculated.

8 Turing, Alan, 'On Computable Numbers, with an Application to the Entscheidungsproblem. A Correction' (1937), in *The Essential Turing*, 94–6. In this note, among other things, Turing specifies an intuitive definition of the term 'computable number'. Without specific mention of Turing, an excellent exploration of the philosophical consequences of the intuitionist concept of number can be found in Evans, Aden, 'The Surd', in *Virtual Mathematics, The Logic of Difference*, ed. Simon Duffy (Bolton: Clinamen Press, 2006), 209–34.

9 Petzold, Charles, *The Annotated Turing* (Indianapolis, IN: Wiley, 2008), 317.

10 See, e.g. Turing, Alan, 'Computing Machinery and Intelligence', in *The Essential Turing*, ed. B. Jack Copeland (Oxford: Clarendon Press, 2004), 433–64.

11 Petzold, *The Annotated Turing*, 307.

12 Ibid.

13 See, Hodges, Andrew, 'Alan Turing and the Turing Machine', in *The Universal Turing Machine, A Half-Century Survey*, ed. Herken, Rolf (Vienna: Springer, 1995), 3–14.

14 van Atten, Markus S. P. R, *Phenomenology of Choice Sequences* (Utrecht: Zeno Institute of Philosophy, 1999), 73.

15 There are other reasons for this prohibition as such programs are also occasionally Trojan Horses.

16 See, e.g., Doctorow, Cory, *For the Win* (London: Harper Voyager, 2010). Castronova, Edward, *Synthetic Worlds* (Chicago: University of Chicago Press), 2005.

17 This project, first performed at the Software Summer School in London in 2000, and subsequently elsewhere, consists of a crowd of people arranging themselves into a grid formation and carrying out a set of instructions inspired by James Conway's 'Game of Life'. See, http://www.spc.org/fuller/projects/the-human-cellular-automata/ (accessed 12.12.2013).

18 Byers, William, *How Mathematicians Think, Using Ambiguity, Contradiction and Paradox to Create Mathematics* (Princeton, NJ: Princeton University Press, 2007), 28.

19 Ibid, 40–1.

20 Brouwer, who thought of language simply as a shed residue of, or impediment to, mathematics, on the one hand, or an exact technique for memorizing mathematical constructions, on the other (see his *Cambridge Lectures on Intuitionism* (1951), (Cambridge: Cambridge University Press, 1981), would probably be appalled at the idea. Edsger Dijkstra, on the other hand, always alert to the medial conditions of thought, suggests, in EWD 1000, that 'No future history of science can ignore the change the advent of the copying machine has made'.

21 Of signal relevance here is the approach exemplified by Bernhard Siegert in 'Cacophony or Communication? Cultural Techniques in German Media Studies', trans. Geoffrey Winthrop-Young, *Grey Room*, 29 (2008): 26–47.

22 For the double bind, see, Bateson, Gregory, Jackson, Don D., Haley, Jay and Weakland, John H. (1956), 'Towards a Theory of Schizophrenia', in *Towards an Ecology of Mind* (Chicago: University of Chicago Press, 2000), 201–27. The 'infernal alternative' is posed by Isabelle Stengers and Phillipe Pignarre in *Capitalist Sorcery, Breaking the Spell*, trans. Andrew Goffey (London: Palgrave Macmillan, 2011).

23 Turing in Queneau, Raymond, *Cent mille milliards de poèmes* (Paris: Gallimard, 1961).

24 Turing, 'Computing Machinery and Intelligence'.

25 Here we can develop a relation to Foucault's understanding of experience. Looking back he reformulates, to some extent, a thematic running through from his earlier to his later work – *experience*. That is to say that 'forms of knowledge, matrixes of forms of behaviour, and the constitution of subjects and modes of being' are experiential, rather than static. Foucault, Michel, *The Government of Self and Others* (Michel Foucault: Lectures at the Collège de France 1982–3), trans. Graham Burchell (London: Palgrave Macmillan, 2010), 5.

26 Beckett, Samuel, 'Quad', in *Collected Shorter Plays* (London: Faber and Faber, 1984), 289–94.

27 Deleuze, Gilles, *Essays Critical and Clinical*, trans. Daniel W. Smith and Michael A. Greco (London: Verso, 1998), 154. See also, Beckett, Samuel, *Quad et autre pièces pour la television suivi de L'épuisé par Gilles Deleuze* (Paris: Éditions de Minuit, 1999).

28 Reich, Steve, 'Drumming' (1970–191).

29 Guattari, Félix, *Chaosmosis, an Ethico-Aesthetic Paradigm,* trans. Paul Bains

and Julian Pefanis (Sydney: Power Institute, 1995), 111. See, for further discussion of such an approach, Guattari, Félix, *The Anti-Oedipus Papers* (Los Angeles: Semiotext(e), 2006) and the riddling of reality with such machines in *Anti-Oedipus*: Deleuze, Gilles and Guattari, Félix, *Anti-Oedipus, Capitalism and Schizophrenia*, trans. Robert Hurley, Mark Seem and Helen R. Lane (London: Athlone, 1984).

30 The term, part of a general discussion in Object Relations theory, runs through much of Klein's work. See, e.g., Mélanie Klein, 'Some Theoretical Conclusions Regarding the Emotional Life of the Infant', in *Envy and Gratitude and Other Works 1946–1963* (London: Hogarth Press and the Institute of Psycho-Analysis, 1975).

31 Klosky, Claude, 'The First Thousand Numbers in Alphabetical Order', in *Against Expression, an Anthology of Conceptual Writingi,* (eds) Craig Dworkin and Kenneth Goldsmith (Chicago: Northwestern University Press, 2011), 148–60. It is notable that this piece of work would change each time it is translated into another language as the alphabetical ordering of the letters would differ, rendering the formalism that generates the work, as given in its title, fundamentally playful.

32 Dworkin, Craig, *Parse* (Berkeley, CA: Atelos, 2008).

33 For a description of the use of Redstone to create circuits, see: http://www.minecraftwiki.net/wiki/ (accessed 12.12.2013).

34 For video of the first ALU, see: http://www.youtube.com/watch?v=LGkkyKZVzug; for initial documentation of the CPU, see: http://www.youtube.com/watch?v=sybOqi_dgX0&feature=related. The program counter and a revised ALU (of 11 October 2010) is shown at: http://www.youtube.com/watch?v=sybOqi_dgX0&feature=related (all accessed 12.12.2013). The cheerful and laconically enthusiastic tone of the commentary by theinternetftw make these a pleasure to watch. The full computer seems not to have been built, however a number of others have since developed *Minecraft* computers of various sorts, including, in June 2011, a dual core CPU by anomalouscobra and jomeister15, see: http://www.youtube.com/watch?v=EaWo68CWWGM&feature=related/, and in September 2011 a full gaming computer *Redgame* by laurensweyn, a video of which is at: http://www.youtube.com/watch?v=lB684ym3QY4&feature=related (all accessed 12.12.2013).

35 von Neumann, John, *First Draft of a Report on the EDVAC*, Contract No.W-670-ORD-4926, between the US Army Ordnance Department and the University of Pennsylvania (University of Pennsylvania, Moore School of Electrical Engineering, 30 June 1945).

36 See Kitchin, Rob and Martin Dodge, *Code/Space, Software and Everyday Life* (Cambridge: MIT Press, 2011).

37 The slight naffness of *Minecraft*, its clunky aesthetics, the laboriousness of the construction work within it, alongside the exuberance of imagination involved and expended in it by millions of players is a subtext of the hugely popular Yogcast series of comedic, lackadaisical commentary on *Minecraft*, posted on YouTube. In the last few years, since *Minecraft* has come out of beta, a highly active scene of 'Let's Play' video makers has emerged with their own channels.

38 Agre, Philip, *Computing and Human Experience* (Cambridge: Cambridge University Press, 1997), xi.

39 Ibid., 13.

Do Algorithms Have Fun? On Completion, Indeterminacy and Autonomy in Computation

Luciana Parisi and M. Beatrice Fazi

Algorithms, not androids

In 1968 Philip K. Dick wondered whether androids, which at the time epitomized the ultimate mechanization of mental processes, could dream of electric sheep.[1] Dick's provocative suggestion was meant to address the prospect that mechanized thought could develop a human unconscious, and thus become truly autonomous. In Dick's novel, dreaming stands as the ultimate expression of the human capacity to think beyond logical processing, without the functions of cognitive reflection or recognition. It could be argued that, in Dick's story, a mechanized being that is able to dream realizes the promise of total autonomy from instructions and pre-programmed finalities. A dreaming android never becomes a human subject, but is nonetheless as close to the human emotional dimension of thinking as it could ever be.

Today, nearly 50 years after Dick's proposition, it is similarly possible to wonder whether algorithms – which are now the quintessence of mechanized thought – could exhibit a dimension of autonomous thinking. While androids had a body but were looking for a soul, in the present day mechanized thought is instantiated in computational processes that are themselves already material and incarnated, insofar as they are always performed and embedded in society, culture and the economy. To be autonomous, these computational systems do not need to simulate the more or less conscious requirements of a 'psyche', insofar as they already have a 'point of view' (or a sort of subjective dimension, so to speak) that is expressed through their *material agency*. Within the current configuration of digital technology, it could then be claimed that the mechanization of thought carried out by algorithms has approached the ultimate goal of total automation by incorporating thinking as material performativity, thus showing an immediate tendency to break free from the human master that has nonetheless programmed them.

Following on these considerations, one could contend that, in computational science and in computational culture alike, the final goal for automated thought is no longer to escape the pre-programmed algorithmic rule and thus to revert into a sort of computational unconscious. Instead, automated thought is epitomized in a new vision of the algorithmic rule itself. Now this rule is always executed, and in this execution it can adjust itself to external changing conditions. To sum up, most algorithms have become *interactive*, and the mechanization of thought has to be accounted for *vis-à-vis* the now dominant expansion of such interactive approaches to computing. It is in this respect, one could argue, that today the quest for the autonomy of mechanized thought involves the prospect of mechanized systems changing their own pre-programmed regulations. This interactive vision of software has importantly questioned the self-sufficiency of axiomatics, logic and formalism, and has tried to complement the formal rule with the environmental input. Consequently, in the cultural analysis of these new modes of mechanization of thought, interactive software has been seen as operating *affectively*; that is, through the expansion of the field of potential activities that disrupt and re-invent meaning.[2] Here, digital technology becomes the ground for creativity and for the production of paradoxical propositions that do not aim to reproduce a pre-constituted intelligence, established by a representational conception of thought. Rather, paradoxes and ambiguities strive to expose the sensible dimension of thought against the dominance of computational understandings of cognition.[3] From this perspective, freeing the computational unconscious becomes secondary to the possibility of producing, in computation, *sense* beyond formal logic.[4]

It is exactly within this context, we believe, that the question of fun in software has acquired a significant momentum. Fun in computing emerges as an affective force that exceeds the formal logic that has nonetheless generated it; fun thus coincides with the disruption, caused by uncertainty, of the decisional power of algorithms operating within changing data structures. We may thus agree that the strength of this proposition is that it offers a radical account of the exceptional condition of instability and malleability of the computational rule – a condition that is vital to the creative drives of programming cultures. To a certain extent, the present chapter builds upon the rich debates that have put forward this proposition, and at the same time it aims to push this new view of computation in another direction.

The question that we propose – 'Do algorithms have fun?' – is meant to echo the hypothesis posed by the title of Dick's novel, *Do Androids Dream of Electric Sheep?* Like Dick's provocation, our own hypothesis may also sound absurd; yet such an absurd proposition may nonetheless provide an opportunity to investigate another scenario regarding the significance of fun in software. To put it in other words, our proposition still aims to question computation, logic and formalism as representational approaches to thought. However, in order to do so, it is necessary for us to revisit the possibility that autonomy in computation might not be geared uniquely towards the liberation of the algorithmic procedure from the program or the master programmer. The question, we contend, is less whether a procedure might become free from mechanization, but whether mechanization itself already comes with a form of autonomy, insofar as it can be defined as a series of procedures aiming towards *completion*. As will become clear further below in the chapter, the notion of completion is used to define the process by which a final determination is achieved.

Algorithmic autonomy for us is not about a machine sharing or opposing the human spectrum of cognition and perception in which the possibility of having fun is traditionally inscribed. On the contrary, the question, 'Do algorithms have fun?', is predicated upon a speculative move suggesting that *fun corresponds to the capacity of algorithms to enjoy themselves while processing data.* Enjoyment here is, in turn, mainly defined in terms of the *final purpose* of the computational process, which is to say, by its functionalist imperative to complete a task. In a counter-intuitive twist of the debate, we therefore take fun not to be the subversion of a rule, or the break from the constriction of a cold and strict procedure, rather we suggest that fun requires an end activity on behalf of the rule itself. Rather than escaping the rule, fun thus points to a new understanding of algorithmic order: one that complies with the question of the fulfilment of procedures, or achievement of a result, in computation. Fun in software is for us neither posthuman (there is no an alleged equivalence between human and machine), nor inhuman (there is no opposition between the two). Fun here deploys another level of mechanized processing altogether – a level that is not exclusively preoccupied with human–machine relations or the lack thereof. In this procedural order, relations are secondary to the finality of the rule's function. *Fun, we contend, is the final achievement of autonomy in mechanical thought.*

Forms of process

The questions raised in this chapter are directed towards the speculative re-conceptualization of computational algorithms in terms of actualities, expressed as *forms of process*. We take the latter notion from the mathematician and philosopher Alfred N. Whitehead, who uses it to define the actual organization of data in the world.[5] In our opinion, this Whiteheadian concept is key to the possibility of re-thinking what processes (and, therefore, also computational processes such as algorithms) are. Following the Whiteheadian view, processes can in fact be re-thought not as a continuous movement of variation, but instead in terms of the final achievement of a finite state.

Whitehead employs the notion of 'form of process' to explain the 'final mode of unity in virtue of which there exists stability of aim amid the multiple forms of potentiality, and in virtue of which there exists importance beyond the finite importance for the finite actuality'.[6] In this respect, Whitehead's 'form of process' needs to be investigated *vis-à-vis* a key element in the Whiteheadian philosophy: *actual occasions*. An actual occasion is actual not merely because it is something that is present here in front of us and impresses us (as Hume would have held actuality to be). In Whitehead, actuality is instead a process of *concrescence*, or 'a process in which the universe of many things acquires an individual unity'.[7] This is, for Whitehead, a process of *satisfaction*, or of *self-enjoyment*. In actual occasions, data are organized and unified; when this unity is achieved, the actual occasion is 'satisfied' because it has fulfilled its scope by becoming complete.

It is our contention that Whitehead's philosophy of actuality can contribute to questioning the antagonism between, on the one hand, the conception of *process* (intended as the continuity of variation, forming a whole that is bigger than its parts) and, on the other, the notion of *processing* (intended instead as a self-contained procedure based on already determined and finite parts aggregating into a whole). In Whitehead's oeuvre, these two concepts are not opposed; it could be said, in fact, that both process and processing contribute to the determination of the actual occasion. We thus wish to appropriate Whitehead's philosophy of actuality to go beyond this opposition which seems to pervade the critical debates about computational culture.[8] To this end, we suggest that algorithms can be conceived as discrete actual entities, whose aim is to achieve unity. This achievement can be defined, again in Whiteheadian terms, as *satisfaction* (which is, in Whitehead's words, 'the completion of the actual togetherness of the discrete components')[9] or, from the perspective of this chapter, *fun*. On the one hand, an algorithm is a procedure that performs a processing, involving the execution of a sequential number of steps that organize data towards a result. This processing, however, does not simply lead to the aggregation of parts into a whole, but rather corresponds to the creation of a new unity that is added upon (and yet is not contained by) the aggregation of parts. Enjoyment, in this respect, involves the processing of data towards a final aim (this aim being the completion of the procedure). On the other hand, however, we want to stress that an algorithm is also a process, inasmuch as it involves an indeterminate capacity for variation. Yet, what should be emphasized here is that this variation is not the result of continuous change, but it is the final product of the process of determination carried out by the algorithmic procedure itself. This process of determination defines an algorithm that attains its completion by realizing its function. Satisfaction, from this perspective, is the exhaustion of the algorithmic function in the process. In computational culture, to sum up, we find evidence that process and processing are defined less by their differences than by a common finality, and that fun (re-worked speculatively via the Whiteheadian notions of enjoyment and satisfaction) might be an instance of such a finality.

To carry out such re-articulation of algorithms in terms of forms of process – or, it is worth repeating here, in terms of discrete entities achieving satisfaction through the unification of data – we need to linger a little longer on the proposed parallel between algorithms and actual occasions. As already mentioned, Whitehead's philosophy of actualities is useful for our investigation of algorithmic computation precisely because his approach is based on an atomistic conception of the universe that might counterintuitively help us to revise the conceptualization of digital procedures. Moreover, it can also help us to re-think what the unity of discrete entities might be from the standpoint of a process of determination. Instead of describing finite entities as already constituted unities, Whitehead argues that actual occasions are the result of a tendency of the multiple to become united. The accomplishment of unity marks for Whitehead the end point of an actual occasion. In this respect, actual occasions are finite and determined, also because they are short-lived. Once they reach unity, their process ends and they perish. One can also add here that completion is the finality of an actual occasion, insofar as its process of unification is its ultimate goal. The atomicity of an actual occasion corresponds to its finality: discreteness is not given, but produced from within the process. An actual occasion, therefore, does not come from nowhere. It inherits data from past actualities and is characterized by the tendency to eventually terminate just such a process of selection that ultimately constitutes it as one. Data, in turn, do not correspond to what already exists. For Whitehead, data are transmitted from the past to the present in a manner that is not simply re-used, but re-processed by the actual occasions under new conditions.

It is important here to stress that for Whitehead this principle of *inheritance* cannot explain, on its own, the 'becoming one' of the actual occasion. Whitehead's metaphysical proposition instead gives a major role to what he calls *eternal objects*. These are a priori yet immanent ideas, which according to Whitehead can be described 'only in terms of [their] potentiality for "ingression" into the becoming of actual entities'.[10] In this sense, an eternal object is *pure potentiality* from the standpoint of the actualities that select them. Eternal objects, then, cannot be described in terms of one potential plane of continual differentiation (for instance Deleuze and Guattari's plane of immanence and/or the *virtual*) or in terms of a plurality of equivalent ideas (such as the postmodern claim about the relativity of truth). Rather, this other element in Whitehead's metaphysics offers us a notion of *indeterminacy*. Eternal objects are in fact non-connected among themselves, and only acquire togetherness when selected by actual occasions. In this togetherness, they concretize indeterminacy by allowing the actual occasion to achieve finality. Eternal objects are maximally unknowable, but nonetheless they inform the determinate state or the atomism of actual occasions.

At this point we should add that both eternal objects and existing actual data are not passively received but are instead *prehended* by the actual occasion. Contra the empiricist perceptual capacity to synthesize the world through sense impressions, Whitehead argues that prehensions are modes of evaluation and selection, measuring and dividing an infinite variety of data each time. Instead of being just sensations, prehensions, for Whitehead, are both *physical* and *conceptual*. This is because each actual occasion 'is essentially bipolar';[11] in other words it has a *physical* and *mental pole*, expressing the actual occasion's capacity to physically and conceptually process data. The physically prehensive nature of an actual occasion means that actual occasions themselves are always informed by what is no longer there, by the data of the past actual occasions that are accumulated and crystallized in any present actuality. Moreover, as we have just discussed, actual occasions are also informed by what is not there at all: the unknown or indeterminate ideas or eternal objects. The prehensions of eternal objects are, for Whitehead, conceptual prehensions. As we saw earlier, eternal objects define the direction towards the determinateness of the actual occasion, its final unity and discreteness. Here again we find the processual nature of actual occasions, in which the indeterminacy of eternal objects adds a new level of determination to the data selected. A process becomes, however, a processing to the extent that data are compressed and arranged in a finite procedure, oriented by the actual occasion to drive towards its own satisfaction. In conclusion, prehensions, we could argue, are the catalysts for the concretization of the indeterminacy of the eternal object. More interestingly, however, they also reveal the persistent concreteness of indeterminacy itself in actuality.

Completion and indeterminacy in algorithmic processing

Having introduced these key elements and conceptual manoeuvres of Whitehead's philosophy, we can now return to this chapter's initial contention: the possibility of re-conceiving algorithms in terms of Whitehead's actual occasions by drawing on his notion of 'form of process'. In our view, to claim that an algorithm is an actual occasion implies arguing that the computational procedure is more than simply processing; i.e. more than a procedure based on pre-established instructions. In this respect, we want to argue that an algorithmic procedure cannot be thought of away from the indeterminacy that it is always tending towards. This passage is complicated, and to develop it we need first to explain what an algorithm in computation is, and what it is supposed to do.

In formal sciences, an algorithm is described as a sequence of instructions meant to fulfil a task. An algorithm is a procedural method for the transformation of input into output; its instructions are well defined and executed through a finite number of inferential steps. This is the understanding of the algorithm that entered computing via the seminal work of mathematician Alan Turing. In his famous paper 'On Computable Numbers, with an Application to the Entscheidungsproblem' (1936)[249] Turing sets out to answer David Hilbert's question of whether, given a certain input, a universal method for taking a yes-no decision could be ultimately found in mathematics. In that paper, Turing rephrased the problem of decidability from the standpoint of computability. The notion of computability, in turn, was defined by Turing in terms of an effective procedure (i.e. an algorithm) to solve a problem in a finite number of sequential steps. In his attempt to answer Hilbert's challenge, Turing developed a simple thought experiment (the Turing Machine) that was meant to mechanize a valid and determinate method for calculation. This hypothetical device performs computations by moving its head left and right along an infinite tape, that is in turn divided into discrete cells; the head moves one cell at a time, writing a symbol from a finite alphabet for each cell according to some given instructions. By proving that certain functions could not be computed by such a hypothetical machine, Turing demonstrated that there is not a method of ultimate decision of the kind that Hilbert had wished for. In addition to this, however, the Turing Machine also offered a viable formalization of a mechanical procedure. Computational processing, as we know it today, was born.

The strength of Turing's proposition is that, via his formalization of the Turing Machine, the function of computation came to be defined as being much more than just that of crunching numbers. Turing's computing machines – and indeed the contemporary electronic digital computing machine that has developed from them – can solve problems, take decisions and fulfil tasks, with the only proviso that these problems, decisions and tasks are formalized through symbols and a set of discrete and finite sequential steps. In this respect, Turing's effort can be rightfully inscribed into a long series of attempts in the history of thought geared towards the mechanization of reason. Turing's foundational work in the theory of computability, and his conception of a computing machine, were meant to put the intuitive and informal operations of a person actually computing a number into formal terms. Mechanization implied that these formal terms could be automatically repeated because they were inscribed as symbols and discretized as a sequence of steps.

At the technical level, we could argue that the Turing Machine epitomizes a mechanism that works insofar as it completes an operation. Moving now to the conceptual level, we can understand this operativity as one tending towards unity. An algorithmic procedure can in fact be seen as an aggregation of parts; when these parts achieve unity (that is, when the task is fulfilled), the mechanical process shuts down and then another procedure starts over again. Here one could object that, in light of what we have just described, the parallelism between an algorithm and an actual occasion becomes a difficult claim to sustain. It could in fact be argued that the Whiteheadian actual occasion does not merely aggregate into a bigger whole; rather, its parts are realized to become another part that is a new unity. In other words, in an actual occasion parts transform themselves in relation to a whole. However, the actual occasion remains, atomic: its parts never complete the universe of possibilities of what the actual occasion could be, or will be. An actual occasion is always a spatio-temporal relation between elements in process; it is indeed an occasion, its own eventuality determines its own prehensions, and, consequently, its own constitution. Continuing the objection, it could be also claimed that, in Turing's formulation, an algorithm, conversely, has to do with pre-established functions (indeed, with axioms). In a Turing Machine, everything is pre-set, pre-programmed – in total indifference to content and context. From this standpoint, a Turing Machine merely performs processing: a linear procedure of cause and effect where one step inferentially determines the other. The objection according to which the parallelism between an algorithm and an actual occasion is flawed would thus seem to be justified: while an actual occasion is dynamic, an algorithm is not.

Having identified these difficulties, might it still be possible to advance our intended parallel between algorithms and actual occasions? Our answer is yes. The parallel between algorithms and actual occasions is still valid by virtue of an inherent and unavoidable character of computation itself: *incomputability*. With his 1936 paper, Turing formalized the notion of computability; however, what is most striking about that proposition is that the formal foundations of this notion were laid on the logical discovery that some things cannot be computed. In 1931, Kurt Gödel proved the fundamental incompleteness of formal axiomatic systems.[12] In 1936, Turing built on Gödel's discovery to demonstrate that computation, in its mechanized formalization, is also intrinsically limited, and that these limitations are due to the formal axiomatic nature of computational mechanisms. Algorithms drive towards completion; this is of course evident at the level of their execution, but it can also be suggested that, at the formal level, completion is always entailed within the finality of the procedure. However, in his logico-mathematical formulation of computability, Turing foresaw that, despite the incredible versatility and efficiency of his proposed calculating machines, there are problems that cannot be solved, tasks that cannot be fulfilled and decisions that cannot be taken through them. These are incomputable problems for which there is no algorithmic decision. A Turing Machine can compute anything that can be calculated via algorithmic means. If something cannot be computed, then nor can it be put in algorithmic terms. For some functions, there is not a yes-no (or binary) answer. In this respect, *completion in computation cannot always be attained.*

After this brief but crucial overview of the logico-mathematical origin of computation, the significance of the proposed parallel between algorithms and actual occasions might perhaps become more evident, and we can thus begin to justify our understanding of computational procedures in terms of forms of process. We want to argue that, while the unknown dimension of the actual occasion is the eternal object, unknowability in computation is expressed through the notion of the incomputable. *We are thus here contending that the incomputable is to the algorithm what the eternal object is to the actual occasion.* It is true to say that the algorithmic procedure is pre-established; it is undeniably a processing of data. However, this processing is an actuality that also has to confront the indeterminacy of the incomputable in the same way as the actual occasion is a finite spatio-temporality whose determination results from a process of prehension of the indeterminacy of eternal objects. In other words, eternal objects are ingredients in the constitution of actual occasions, and thus their indeterminacy becomes one with the finite atomistic nature of the universe. Similarly, the unknown condition of an algorithmic occasion is an *incomputable condition*, which serves the algorithmic actual occasion to speculatively become more than it was.

The work of mathematician Gregory Chaitin can now be introduced to further expand on the contention that the incomputable is the eternal object in computational processing. In his algorithmic information theory, Chaitin combines Turing's computability with Shannon's information theory to understand computational processing in terms of calculation of probabilities – probabilities that aim to include what in this processing cannot be known in advance. Chaitin understands the threshold of computability from the standpoint of maximally unknowable probabilities.[13] In every computational process, he explains, the output is always bigger than the input. For Chaitin, something happens in the computational processing of data, something that challenges the equivalence between input and output and thus the very idea that processing always leads to an already pre-programmed result. This something, according to Chaitin, is *algorithmic randomness*. The notion of algorithmic randomness implies that information cannot be compressed into a smaller program, insofar as an entropic transformation of data occurs between the input and the output of algorithmic processing, resulting in a tendency of these data to increase in size. From this standpoint, the output of the processing does not correspond to the input instructions; its volume tends, in fact, to become bigger than it was at the start of the computation. Chaitin has explained the discovery of algorithmic randomness in computational processing in terms of the incomputable: increasing yet unknown quantities of data that characterize information processing.

Space constraints prevent us from expanding further on Chaitin's work; the brief overview above should, however, suffice to sustain the claim that his work is indeed useful to us, inasmuch as it can contribute towards explaining what we mean by the 'form of process' in computation. Chaitin's investigation of the incomputable reveals, in fact, that the linear order of sequential procedures (namely, what constitutes computational processing) shows an entropic tendency to add more parts to the existing aggregation of instructions established at the input. Since now this processing inevitably includes, from the perspective proposed by Chaitin, not only a transformation of existing data, but also the addition of new data on top of what was already pre-established in the computational procedure, we believe it becomes possible to argue for a form of process in computation. In other words, here processing is also a process. From this point of view, computational processing does not guarantee the return to initial conditions, and does not simply correspond to the aggregation or disaggregation of parts that can be pre-programmed. This is because Chaitin's conception of incomputability no longer perfectly matches with the notion of the limit in computation (i.e. the limit to what is calculable). Chaitin's incomputable involves, in fact, the addition of new and maximally unknowable parts to the whole; parts, however, that might be bigger than the aggregate whole. For us, such a reworking of the incomputable is striking and speculatively productive, because what was conceived to be the external limit of computation (i.e. the incomputable) has now become internalized in the sequential arrangement of algorithms (randomness works within algorithmic procedures). One can thus even suggest that algorithmic randomness is not 'outside', but has become constitutive of the actuality of the procedure. Processing then includes a form of process, because indeterminacy is a fundamental part of information.

But if the incomputable involves parts that can be bigger than a whole, how do we explain the completion or the achievement of a task of an algorithmic procedure? Isn't algorithmic randomness (i.e. the tendency of information to increase in size) always there to threaten any achievement of a function to complete a task? This is where, by Chaitin's own admission, it is necessary to see algorithmic randomness as a continuation of Turing's attempt to account for indeterminacy. Whereas for Turing some tasks cannot be achieved, and thus computation stops when the incomputable begins, for Chaitin, all tasks have a margin of incomputability. In Chaitin, incomputability does not break from completion. A closer investigation of his theories helps to reveal that algorithmic randomness has changed the nature of computational processing:

completion has been expanded beyond its limits to involve the processing of maximally unknown parts in order to accomplish a task. To put it in other terms, completion is now demarcated by an indeterminate condition that subtends all efforts equating computational processing with the calculation of pre-programmed and already known outputs. In Chaitin, the finality of completing a task importantly requires that instructions respond and resolve unknowns each time the indeterminacy of data emerges in processing.

Interactive procedures

At this stage in our discussion, it may not be too difficult to justify the question of indeterminacy as being a speculative problem within computational processing itself, and thus point to the existence of a 'philosophy of the algorithm' that should attempt such speculations. We started our own speculative investigation by proposing a parallelism between Whitehead's notion of actual occasions and the computational notion of completion. Having discussed incomputability via Turing and Chaitin, it became possible to think that the algorithmic completion of a task cannot be achieved without the participation of indeterminate quantities of data within processing. In particular, we have sought to develop Chaitin's mathematical work on algorithmic randomness to prove that indeterminacy is always part and parcel of a determinate actual occasion/algorithm as it strives towards completion. From this perspective, we believe that completion in computation corresponds to unity, and unity is, in turn, a terminal point that can be reached only if indeterminacy ingresses the processing of data and makes the algorithmic procedure both a process and a processing. In other words, the 'terminus' of an algorithm can be approached only if the incomputable partakes in its procedure.

Recapitulating, we can think of the incomputable as the indeterminacy that is necessary for completion, or for the unity of a form of process. However, our revisiting of computation through the problems of completion and indeterminacy cannot be fully grasped without discussing the dominance of a new paradigm within computing: the interactive paradigm.

Arguably, interaction has become an all-encompassing principle of explanation in contemporary technoscience. Fields of enquiry such as biology, social theory, neuroscience and robotics have all pushed forward an alternative understanding of thought – understandings that see the latter as always already environmentally 'situated'. Interaction is, in this respect, the *interactive behaviour* of multiple distributed agents. In computing, prediction and simulation now have to account for situations that are outside the confines of the black box and are instead always embedded into and dependent upon the environment. This interactive approach has clashed with the algorithmic constraints of the Turing Machine. Turing's model of computation has been considered insufficient or unable to cope with the complexity of the empirical world – a complexity that, one could say, philosophically speaking, has its own non-representational logic. In this respect, efforts have been advanced in computer science to revisit Turing's algorithmic modelling.[14] While Turing's conceptualization of mechanism based on *a priori* instructions can arguably be said to correspond to first-order cybernetics (due to its closed system of feedback), the combination of environmental inputs and a posteriori instructions proposed by the interactive paradigm more clearly embraces second-order cybernetics and its open-feedback mechanisms.

Returning to our investigation of computational completion, we can claim that what we defined earlier as the algorithmic drive towards completion has perhaps changed now. In particular, we want to stress that if interaction is more powerful than algorithms,[15] it is because indeterminacy, from the point of view of interaction, might now imply and represent something else. The volatility and malleability of lived situations are no longer obstacles for mechanisms of prediction. The goal of interaction is indeed to account for variation and novelty, and to enlarge the horizon of calculation to include qualitative factors as external variables of the mechanism. We want, thus, to suggest that, according to an 'interactivist' approach, the problem of the incomputable might be partly eluded. This is because completeness is no longer a solipsistic affair; rather, completeness becomes a prospect that is achievable only by virtue of the contribution of the outside world. In this sense, the algorithmic procedure might be incomplete *per se*. However, it reaches completeness by virtue of its interactive execution. From the standpoint of interaction, then, the successful running of an algorithm is a performance *in the environment* (i.e. computation is embedded in the world) and *of the environment* (i.e. computation needs the world and the data extracted from it to fulfil the algorithmic task).

In our opinion, various approaches to interactive computing share a common goal of pointing towards a new mode of mechanization of a procedure, in which the starting condition of the program does not dictate the procedure's final output. We can see this mode everywhere, from the imperative of participation in artworks, to social media strategies for user input geared to data-mining via neuro-cognitive mappings that now involve actions and perceptions. Taking into account such ubiquity and pervasiveness of computational interaction, it should be commented here that interactive computing is the result of the challenges that computation has to confront today: a quick and efficient responsiveness to vast data spaces, the quantification of desires, beliefs, inclination and knowledges that underpins the statistical calculation of trades and prices. We are keen to stress that, in this contemporary scenario of computational interactivity, completion is attained by adding new levels of quantification of variation. These new levels afford completeness insofar as the function of the algorithm is extended by external inputs, and is, thus, able to bypass its internal limits by simply posing the limit of computation elsewhere. The interactive paradigm, then, concerns the capacity of algorithms to respond and adapt to its external inputs. However, we want to suggest that this interactive form of adaptation does not serve to overcome, but only to postpone the threat of computational indeterminacy.

In order to understand further how these new dynamic attributes of the interactive algorithm work, we can ask whether the interactive algorithm could be seen as being conceptually closer to Whitehead's actual occasion. Is this discussion of interaction in computation a way to prove our parallel between the two notions? The answer is no: we believe that there is an inherent problem with the interactive paradigm's reworking of indeterminacy aiming to calculate the variations of lived situations. Indeterminacy, we argue, is only approximated by the interactive algorithm; the problem of the incomputable is not solved but merely optimized through the addition of an external input. In this sense, the interactive algorithm harnesses the speculative power of the incomputable for computational processing. Going back to Chaitin, we can argue that the incomputable is not simply a limit that must be resolved by the addition of environmental variations, but that – as a problem – it remains an active ingredient or element of every computation, however many variables and data may be involved in it. *Incomputability, like an eternal object, conditions the finality of the algorithmic occasion.*

We are thus saying the interactive paradigm is not so different from the Turing one: both are geared towards the fulfilment of a task. From both perspectives the pre-established procedure clashes with the unknown quantities of the incomputable. While in Turing computation stops, with the interactive algorithm the problem is bypassed by the addition of more and more data so as to enrich, diversify and vary the goal of the function. It could then be argued that the interactive paradigm interprets the incomputable from a human (or humanist) standpoint: it tries to solve the limit from the viewpoint of the human capacity to comprehend or compress data, or as how a human would cognitively cope with the unknown. Instead, our contention is that in fulfilling a task the algorithmic procedure inevitably confronts information indeterminacy, and by this encounter it fully realizes itself. This is completion from the standpoint of a computational process that cannot 'not' be incomplete. In other words, the limit of computation cannot be eluded, as it is intrinsic to computation itself. Computation may account for variations from the environment and thus become more powerful because of this inclusion. Yet the internal limit remains as the mark of the radical indeterminacy that constitutes computation.

Satisfaction and the autonomy of rules

The question of confronting indeterminacy and fully realizing oneself in relation to it needs to be explained by returning yet again to Whitehead. Whitehead, we said, calls the capacity of an actual occasion to constitute itself through the prehensions of data coming from other actual occasions and eternal objects 'satisfaction'. An actual occasion reaches satisfaction as it reaches completion. For Whitehead, this condition of 'exhaustion' corresponds to the constitutive function of every actual occasion, and is not simply an option. Satisfaction should thus be interpreted not as emotional gratification, but as the final fulfilment of an appetite. In this sense, the actual occasion's tendency to reach completion corresponds to its final determinations, which, however, as we have seen, can only be achieved through the ingression of indeterminacy in actuality.

It is exactly at this point in our discussion that we would like to go back to the debate about fun in software. The long treatment of algorithms as actual occasions was in our opinion necessary to add an ulterior challenge to the conceptualization of fun in computational culture. Here, we return to our initial question: 'Do algorithms have fun?' So far we have proposed that we can articulate this question in the Whiteheadian terms of satisfaction and enjoyment, and, thus, in relation to the question of completion in computation. *An algorithm that has fun is an algorithm that 'enjoys' its own process of determination.* As anticipated earlier, in computation this process of determination is explicated in the processing of data on behalf of sequences of instructions. Drawing on Chaitin, we have said above that this processing involves a tendency of information to increase in size (i.e. algorithmic randomness). Via Chaitin, it is also possible to argue that the introduction of randomness, entropy or indeterminacy within computational processing describes algorithms that select a multiplicity of data which gets unified into an output. This output is a unity that is still a part, but that, however, can be bigger than the whole set of instructions from which the computation started. From this standpoint, and yet again following Whitehead, we can add that satisfaction can only correspond to the final expression of such constitutive indeterminacy within computational processing. Here, one can see that satisfaction, understood as algorithmic completion, might bring a further level of speculation to the debate about fun in software. From this perspective, there is fun in software because the algorithmic procedure entails a dimension of enjoyment that derives not from the breaking down of the mechanization, but instead from the fulfilment of the internal dynamic of completion via indeterminacy in computation. We do not deny that there is fun in going outside of the grains of formal logic, as the epitome of representational thought in computing. We maintain, however, that computational processing already implies a dimension of enjoyment or satisfaction – one that is being understood here as a process of completion determined by the ingression of indeterminacy within algorithmic procedures.

Despite the fact that the process by which an algorithm enjoys itself might still remain obscure, we believe that this proposition offers an opportunity to address one of the main challenges that software poses. Crucial to the history of computation is the quest for an ultimate procedure for the mechanization of thought, and thus the possibility of finding a new form of conceptual function. To propose that algorithms are actual occasions means to confront both this quest and this possibility, and to reformulate what the mechanization of thought could be from the standpoint of the Whiteheadian notion of 'form of process'. We want to argue that, in a Whiteheadian sense, the algorithmic mechanization of thought does not aim to establish an equivalence between emotional intelligence and the execution of a rule. By taking algorithms and not androids as our object of investigation, we can say that we are ultimately interested in re-directing the question of the autonomy of thought from emotion to reason. As mentioned earlier, we suggest that algorithms, taken as the contemporary epitome of automated thinking, reveal that the execution of rules (indeed, the mechanical) might have another order of autonomy – one that can be found not without or against but within reason. This is because algorithmic reason is already mechanical, insofar as it is the processing of rules.

This last sentence should, however, be introduced with a disclaimer: we are not suggesting that algorithms exhaust all forms of thinking, nor are we asserting that all thought is determined by reason or by the processing of rules. *We are rather making the case for a mechanization of thought that is already a form of reason.* Algorithms might have an autonomy that is not to be understood as mere replication of what makes us autonomous (this is instead what Dick's androids were aiming to do). Algorithms, we believe, have their own form of autonomy: one that pertains, we are keen to stress, to the final aim of executing rules (i.e. completion).

The question of whether algorithms have fun becomes, from this broader reading of the mechanization of thought, an enquiry into instructions that operate by means of decision. It could be argued that our vision of fun in software shares, to a certain extent, some of the assumptions that motivated Dick's critique of mechanical thought. His dreaming androids were meant to think beyond the rule, so as to deviate from what the traditional idea of mechanized thought was based on (i.e. the repetition of a rule and the constant return to initial conditions). Dick was thus exposing the limits of representation to contain the reality of thought. For Dick, a computational unconscious was a way to argue for non-representational thought, or for drives that cannot be reduced to symbols and steps, and which, thus, cannot be mechanized. In this respect, our contention that algorithmic fun needs to be understood in terms of computational completion similarly opposes the idea that rules represent thought. Ultimately, representation is for us, just as it is for Dick, insufficient to explain the reality of a process. The difference between our proposition and Dick's, however, is that while his androids oppose rules to imagination by emphasizing the indeterminacy of dreaming and/or affective thought, our algorithms/actual occasions reveal that procedural and mechanical thought already contains much more indeterminacy than one could imagine. This indeterminacy is not simply readable via representational means, but remains nonetheless logical, formal and computational.

This is the sense in which we understand the mechanization of thought as a rule-based mechanism that does not simply exclude or optimize indeterminacy, but instead cannot avoid expressing it. This is also to say that the incomputable, far from being the ultimate indeterminacy that eludes computation, is instead the indeterminate condition of every information system and, as such, it is constantly realized in algorithmic procedures. Our emphasis on the incomputable *vis-à-vis* Whitehead's eternal objects therefore aims to put forward the idea that something has changed in the nature of the rule itself. We suggest that the computational rule is dynamic, but this dynamism is not uniquely derived from the interaction that the rule entertains with its outside when executed. In our view, the rule is instead dynamic by virtue of its processual function. Computational processing is thus also a process, even though it cannot be reduced to it.

In conclusion, we suggest that *computational processing can correspond to a non-human intelligibility of indeterminacy*. From this perspective, algorithmic rules are what they are: instructions. As such, they are meant to fulfil a task. Yet, we believe that this instrumentality is not a symptom of an undefeatable power that programs our very human capacity of thought. Instead this instrumentality can be understood as defining an irreducible, and fundamentally algorithmic, form of process. In this respect, the fact that the rule accomplishes a function is what for us justifies the autonomy of such a function. To ask whether algorithms have fun is then a speculative exercise to test the hypothesis of the autonomy of algorithms as a new form of mechanical thought. Perhaps the question remains absurd, nevertheless it must be posed.

Notes

1 Dick, Philip K. *Do Androids Dream of Electric Sheep?* (New York: Ballantine Books, 1996).

2 See, among others, Munster, Anna *Materializing New Media. Embodiment in Information Aesthetics* (Hannover: Dartmouth College Press, 2006); Hansen, Mark B. N., *New Philosophy for New Media* (Cambridge, MA: MIT Press, 2004).

3 Such understandings would instead take thought to be a mere aggregation of fixed rules. This occurs, for instance, in Putnam's computational theory of mind: Putnam, Hilary *Mathematics, Matter and Method* (Cambridge: Cambridge University Press, 1979).

4 The notion of 'sense' here is derived from Deleuze's project to challenge the logic of representation from within. Deleuze, Gilles, *The Logic of Sense* (London: Continuum, 2004). Against the legacy of representational thought and theories of signification, Deleuze addresses and overcomes the linguistic turn in philosophy by proposing a 'logic of sense', according to which the genesis of value and significance is more important than truth and meaning conditions. For Deleuze, sense is neither original nor final, but an ontological event that is expressed in its effects – effects that are intensive, affective, material.

5 See Whitehead, Alfred N. *Modes of Thought* (New York: The Free Press, 1968), in particular, refer to 'Lecture Five'.

6 Whitehead, *Modes of Thought*, 86.

7 Whitehead, Alfred N., *Process and Reality* (New York: The Free Press, 1978), 211.

8 Massumi's elaboration of process and processing in terms of an opposition between the analog and the digital (both conceived as modes of articulating the real) has arguably influenced many juxtapositions of these notions in computational culture. For Massumi, analog processes are ontologically 'superior' to digital processing. The latter, for him, can never fully account for the virtual potential of thought and experience, a potential that is continuous (therefore, analog) and not discrete (or digital). See Massumi, Brian *Parables for the Virtual. Movement, Affect, Sensation* (Durham, NC and London: Duke University Press, 2002). See, specifically, 'On the Superiority of the Analog'.

9 Whitehead, *Process and Reality,* 85.

10 Whitehead, *Process and Reality,* 23.

11 Whitehead, *Process and Reality*, 108.

12 Gödel, Kurt 'On Formally Undecidable Propositions of the *Principia Mathematica* and Related Systems I', in *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions*, ed. Martin Davis, trans. E. Mendelson (Mineola, NY: Dover Publications, 2004), 4–38.

13 See Chaitin, Gregory, *Meta Maths. The Quest for Omega* (London: Atlantic Books, 2005); Chaitin, Gregory, *Exploring Randomness* (London: Springer-Verlag, 2001).

14 For an overview of some of the key issues in interactive computing, refer to Goldin, Dina, Smolka, Scott A. and Wegner, Peter (eds) *Interactive Computation: The New Paradigm* (New York: Springer, 2006).

15 This hypothesis has been advanced, for instance, in Wegner, Peter, 'Why Interaction is More Powerful than Algorithms', *Communications of the ACM*, 40(5) (1997): 80–91.

5

useR!: Aggression, Alterity and Unbound Affects in Statistical Programming

Adrian Mackenzie

How should we think about the agency of programmers? Today, at a moment when data aggregation has become the mantra in so many sectors of science, business, government and entertainment, and when aggressive aggregations of data are in play all around us, it might be useful to explore how programmers embody data structures. Rather than seeing programmers' agency as rooted in the force of algorithmic execution, it might be better to see how data structures bind the psychic energy of programmers into certain forms of movement, feelings of potency and somewhat aggressive and eroticized attachments. If we start from the position that the bodies of programmers are animated by all the aggressions, satisfactions, violences, desires and pleasures characteristic of subjectivity more generally, then we might better understand the alternately excited and enervated feelings associated with bringing software into being. Conversely, if data structures bind psychic energy through the ways that they channel, grid, slice, fold, ply and sort multifarious relations, we can sense how software aggregates and clusters not just data, but realities more generally. To the extent that programmers embody – in habits, in perception, in sensation, in affect, in artefacts – the relational textures of the aggregates they encounter, they in turn replicate and extend the dimensions of that relational matrix. The question is how they energize or animate what they encounter and how they render it inert and moribund.

In any human embodiment, there are polarities between what might loosely be termed aggression and non-destructive eroticization. In the former, to follow a psychoanalytically inflected train of thought, fantasies of rootedness, of foundation, of wholeness and mastery can only be attained by acting as if the world can and must be controlled. This is acted out aggressively. In the latter, the ego or self is susceptible to new associations, encounters and exchanges. Associations and encounters, insofar as they are new or different, take the form of unbound affects, affects that fall outside iteratively maintained psychic organization. Such unbound affects are necessarily eroticized. Yet aggression and eroticization are connected. It is not as if we could just say 'no' to aggression. As Leo Bersani and Adam Phillips suggest, any 'satisfied aggression is accompanied by an erotic excitement'.[1]

The erotic excitement triggered by the hallucinatory process of fantasizing control over the world and others is felt as power, potency and narcissistic jouissance. Its power is aggressive since in separating itself from the world, it also renders the world as harmful. Yet this aggression is not without risk. It risks shattering the ego. While what Teresa Brennan calls 'the foundational fantasy'[2] brings with it feelings of omnipotence and control over the world, these feelings tend to grow hyperbolically. As Bersani and Phillips write: 'the "extraordinary narcissistic enjoyment" that accompanies satisfied aggression at once hyperbolizes the ego and risks shattering its boundaries'.[3] Any flow of energized excitement eroticizes the subject-to-be in ways that are potentially associative, and indeed, sometimes freely associative. This freely associating excitement or unbound affect is not necessarily aggressive or destructive. It may open the possibility of what they call 'non-destructive eroticizing of the ego'.[4] Put in a more political register, it might embody an 'appetite for … fresh forms of conflict, … for an unpredictability of feeling and desire'.[5]

To even use the term eroticization in the context of software might be read as tendentious or far-fetched. Yet this connection is more tenable if you think about the software of eroticization. The most obvious zone of exchange between software and eroticization is probably online pornography. While in this discussion I say little about pornography and software, the tremendously iterative tendencies of contemporary pornographic media could be analysed using software studies approaches. The result of such an analysis might not be too different from what I am endeavouring to discuss here in several respects: how the conservation or maintenance of psychic organization takes place; how the hyperbolically growing effort to maintain existing forms of eroticization cannot ever fully extinguish the risk of shattering or dissolution; and how any attempt to anticipate desire is reliant on yet eluded by the populations, aggregates and multiplicities that vitalize bodies. This entwining of repetition and alterity lies at the heart of any ethopolitics of software. All of this plays out around software. The question is what kind of agency software wants: will it be a new superordinate egoistic power, the epitome of rational will or accepted entitlements, or something that frees and proliferates forms of economic, communicative and erotic exchange?

Embodying data structures: From predictive analytics to data munging

We might begin to understand something of how contemporary programmers embody these polarities through the case of a single programming language, R.[6] R is a programming language, a computing environment and a cluster of code libraries focused on statistics. It was designed almost exclusively for the purpose of collecting, binding and cutting data in support of forms of statistical inference. R is almost 15 years old, roughly the same age as Java or Python. It derives from an earlier statistical programming language and environment, S, that was developed at Bell Labs and commercialized in the late 1980s as S-PLUS. Like UNIX/Linux/GNU, and roughly the same age, R's biography is somewhat entwined with the history of Bell Labs. Although unlike Linux/GNU it languished in the late 1990s, in the last few years R has been reanimated and attracted much more investment in academic work and in commercial settings.[7] As evidenced by the bulging blogroll at R-bloggers.com, R has also become much more than a tool to teach statistics to college students. People are playing with it as a way of working with data much more widely. Both as programming language and as comprehensively networked archive of data analysis packages (approximately 3,000 in CRAN – Comprehensive R Archive Network),[8] R has become a thriving ecology of software data practice. In R, statistical methods, data manipulation tricks, data extraction techniques and code for graphics and visualization rapidly move laterally between diverse fields of high and low sciences, industry, government and media. Code and scripts, often transiting very quickly between research scientists and commercial and non-commercial operational settings, also switch back and forth between media, science and business. No longer is statistics in the service of governmentality or the biopolitics of population, but in the service of entertainment, knowledge and security. R, with its several thousand packages, with its cross-platform builds, with its open source and with its very rapid absorption of the latest advances in statistical technique, data structures and sources of data in many fields (Twitter feeds, GoogleVis, pubmed API, World Bank data, etc.), is part of this shift. The movement of methods is itself one sign of excitement about R.

Through R it might also be possible to think about how software embodies shifting desires, excited beliefs and eroticized enjoyment concerning data. In contrast to languages supporting the design of user interfaces, or languages equipped to grapple with network protocols and infrastructures, or languages designed to deal with the unruly, unwieldy sociality of programmers working together, the strength of R consists in an extraordinary profusion of data structures and techniques for working with data at every level from elementary sorting to advanced statistical modelling. Writing code in R is not really associated with networking, with graphic user interfaces (windows, widgets, controls, etc.) or with building platforms (operating systems, web services, applications, etc.). Rather, statistical programming connects programming practices to flows or sources of data apprehended as samples, experiments, observations, surveys or measurements. While many of the same programming techniques found in Python, Java, Perl or Lisp can be found in R, or applied in R, R is software and programming with a different end: to do things with data statistically. There are many ways of doing things with data: acquiring it, possessing it fully, using it and showing it. Data comes in diverse forms. Most datasets probably exist as Excel spreadsheets. Tremendous amounts of data are in relational database tables. And increasingly, data exists as images or as structured files and streams (XML, RSS, RDF, etc.). For all these forms of data, there are typical practices of editing, curating, viewing or analysis. The question is what is going on with work on data in R? Is there something here that undoes investments in code as control? Wouldn't statistical software be in the service of the taming of chance, as Hacking would put it,[9] or the politics of large numbers?[10]

Every programming style brings with it specific investments in data structures. For instance, structured programming languages such as Pascal and Ada, or 'object-oriented' languages such as C++ or Java, bind data together with specific software architectures and programming practices organized around hierarchical structures. At the same time, these bindings of data, environment and practice are energetically connected to diverse historical enactments. Something of the diverse contemporary investment in data and data structures can be found in R. Different dynamics and logics are mixed in R. In order to filter these, we could explore how R attracts collective attention at two annual conferences, Predictive Analytics World and UseR! At the Predictive Analytics World, San Francisco in February 2009, high-end analytics software vendors – Spotfire, SAS, IBM, Oracle – are on hand alongside the database and cloud services.[11] The 'Encounters with R' session, run by the Bay Area UseR! Group, is full. It seems R's heyday has arrived for the business data miners. In the R session at Predictive Analytics World the panel of speakers come from Google, Facebook and Revolution, a company set up to commercialize a version of the open-source statistics programming language and platform R. The speakers are talking about how they cook the fat data gathered in their respective enterprises. The Google researcher, Bo Cowgill, drifts on rather monotonously about how Google uses 'classifiers' to speed up their search engine offering. Itamar Rosenn from Facebook much more energetically talks about the use of 'logistic regression and recursive partitioning' to model user 'receptiveness', and to help predict who will stay and who will drop off Facebook. He refers to particular packages in R, and says how they used them. The presence of Google and Facebook signals a certain buoyant mood that infuses discussions of R. At Predictive Analytics World and very many other places, data talk is increasingly suffused with a frisson of excitement about doing things to and getting things from data. Data, it seems, is being aggressively pursued. In other settings, this excitement about using R on data is invested slightly differently. Epistemic, aesthetic, economic and political values accompany talk of open data, data journalism, data mashups, infoviz. The final speaker, David Smith from Revolution, voices some of these less hard-headed, business-minded values. Like the founder of Revolution, Norman Nie, who developed the very-well known social science stats package SPSS at the University of Chicago in the 1970s, he evangelizes a souped-up commercial version of R in terms of democraticization:

> Everyone can, with open-source R, afford to know exactly the value of their house, their automobile, their spouse and their children—negative and positive … It's a great power equalizer, a Magna Carta for the devolution of analytic rights.[12]

Interest in business data is nothing new. Alongside nuclear physics and weather modelling, business imperatives to process data have shaped software architectures in many ways; for instance, in the form of relational databases, which were largely conceived in response to business rather than scientific needs.[13] There have long been overlaps and exchanges between these domains. Scientific techniques of simulation have flowed over into commercial settings, particularly via the circuits of military-funded research. Many scientists work in industry and develop software for industry. Conversely, scientists make use of much commercial software, ranging from software such as Weka and Matlab to Oracle databases. The work of computer scientists is particularly difficult to disentangle. Their work runs crazily across the lines between knowing and selling, people and things.

The current R boom is not solely driven by the needs and desires of business intelligence under discussion at Predictive Analytics World and the many other 'analytics' events. The rise of R is no outlier. Big data discourse (especially in business intelligence, market analytics, quantitative trading finance markets and various life, physical and social sciences), the promise of data economy, as well as shifts in governmentality (tied to the erstwhile Big Society in the UK, and in Australia, Canada and the United States with open data initiatives) converge in many ways. The rising star of 'data scientists' in business and government, however, owes much to the diverse ecology of code, data, models, techniques, infrastructures and practices that has grown around R in the last two decades. Data is very much allied to what has been happening in sciences across the spectrum of physical, earth, life and social sciences. Scientific software, made to work with data from instruments, sensors and databases, usually barely becomes visible in popular culture. In fact, with statisticians and the rare computer scientist, it is lowly field scientists such as ecologists, geologists and sociologists who have contributed much to the rampant tropical growth of R. Non-commercial, often patchily architected, slovenly bodies of R code abound. These lower-order, epistemically, aesthetically and economically diverse data intimacies can be seen at the other conference settings, the UseR! Conferences.[14] Since 2004, the UseR! Conference has taken place in Europe and North America. In comparison to Predictive Analytics World, these conferences are largely academic, with a sprinkling of industry participation. While in recent years oddly mixed sponsorship has started to appear from Google, SAS and GCHQ, the general frame of these conferences is academic and technical rather than business. The title of the conference and many local groups of the same name make a small word play: useR! is both an injunction to use the programming language and functions as a mathematical expression of the rapid growth of use of R via the factorial function (R!). The hundreds of presentation in these conferences are factorial in relation to Predictive Analytics World: they address a much wider range of topics ranging from DNA microarray analysis to male lizard fights, from deployment of R on Debian to real-time quantitative analysis of Bloomberg trading data. Scientists and statisticians rather than professional programmers are doing the software development in these settings, but they themselves are highly receptive and responsive to the many shifts in software cultures surrounding them.

The love of munging

Can there be non-destructive enjoyment of data? Given the range of scientific techniques, models and styles mingling in R, can data-driven desire set free different solutions to the conflicts between exhaustion and speed that beset software cultures? Framed from the psychoanalytic perspective, the question would be: can there be pleasure here that is not an erotic excitement of satisfied aggression, or narcissistic data-control jouissance? What kind of other desires might be possible? Deleuze and Guattari argue that 'desire belongs to the infrastructure'.[15] Their target is ideological understandings of desire that locate it in the mind of subjects. They seek instead to discover in the subject 'the nature, the formation, or the functioning of *his* desiring-machines, independently of any interpretations'.[16] The desire that belongs to the infrastructure: might that be more like the non-destructive eroticization that Bersani and Phillips speak of? Could R, and the cuts it makes in data, undo anything of the flows that carry coding towards the hyperbolized ego of Predictive Analytics World?

Much analysis of software has focused on algorithms as control processes or protocols.[17] Data structures – lists, arrays, tables, sets, graphs, databases – have attracted attention either as media forms or in relation to control processes.[18] In some ways, statistical programming falls somewhere else. It incorporates various styles of coding, different kinds of programming tools, software architectures and bodies of code. It moves between extremes of infrastructure, from spreadsheets to dense scientific or business computing clusters. However, it incorporates biopolitical aspects which are less visible in other programming settings. By virtue of being statistical, R is connected to the several centuries of scientific-governmental-institutional investment in survey, sampling, randomizing, testing and plotting. This biopolitical underpinning of R affects it at every level, from the texture of code written in R to the way R is taken up, affirmed or validated in various settings. Work at either extreme of data – the mundane collecting, filtering, reshaping, transforming, transposing, sorting and ranking that makes things into tables, lists and files, and on the apparatus of probability distributions and statistical models which many data-mining algorithms depend on – figures much less. For the prosaic work, there is term 'data munging'. More scientifically prestigious work is epitomized in the field of statistical machine learning, where clustering, classifying and modelling are exhaustively discussed.[19] Yet working 'data scientists' (a term that seems to have sprung into existence in the last few years) commonly estimate that 80–90 per cent of their work is getting data into the right form for analysis. Data comes from scenes of much intensely focused 'munging',[20] but working with data takes software in different directions, especially in its encounter with populations.

There are good grounds for thinking that much of the energy going into R is stultifying if not stupefying. Take the case of Google's chief economist, Hal Varian. On the one hand, Varian is widely quoted by proponents of R for his views on data science:

> The ability to take data—to be able to understand it, to process it, to extract value from it, to visualize it, to communicate it—that's going to be a hugely important skill in the next decades, not only at the professional level but even at the educational level for elementary school kids, for high school kids, for college kids. Because now we really do have essentially free and ubiquitous data. So the complimentary [sic] scarce factor is the ability to understand that data and extract value from it.[21]

This immediately raises the question: important for what? On the other hand, Varian's own offerings in this area are not terribly inspiring. For example, he dips into the mighty data torrents of Google search and GoogleTrends, channels them through some relatively minimal R code that uses seasonal autoregression (a technique developed in econometrics to measure correlations in fluctuating prices over time) in order to leverage a few per cent increase in the confidence with which US car sales or tourist travel to Hong Kong can be predicted 'in the present'.[22] The techniques, the code and the predictive results are hardly startling. Nonetheless, a couple of things might be indexed by Varian and Choi's work on GoogleTrends using R. First of all, this is an example of R scripts bringing scientific techniques into play in the service of predictive analytics. Second, the empirical data to which the statistical models fit is itself produced by other layers of software that draw out and channel flows of desire. The flow of Google searches are, as is well-known, the raw materials for the analytics-driven advertising businesses of Google worlds. In these settings, production of surplus value relies on making sense of what people are doing. Cutting these flows, the R scripts are desiring-machines. The question is: how far can the techniques, scientific models – techniques and knowledges that really come from an incredibly wide range of angles – cut into the flows of data generated by an axiomatic machinery of predictive flows? We can see both a concerted attempt to apply the statistical methods developed in the last 200 years in life and human sciences to new terrains, to new populations. The principle that

Varian and Choi's work proves is that desires and beliefs can themselves begin to be treated as populations whose parameters and distribution can be estimated.

While Google's use of R to 'predict the present' seems like a neat proof of principle, the Facebook case is more provocative. In his Predictive Analytics World presentation, the Facebook researcher Itamar Rosenn described how Facebook attempts to mine its user data using R. No surprise there: any business faced with hundreds of millions of registered users might want to try to find patterns or regularities in how people use their products. Facebook's interest lies in trying to predict whether people will come back to Facebook after registration. Although more than 500 million people have registered, it is hard to say how many people actually do anything on Facebook's pages. Their commercial existence depends on user retention. However, despite the claims of data-mining software vendors, making sense of such vast data sets is no simple operation to be executed according to a rational plan. If it was, Facebook would not be using R. A turn to R, rather than to an IBM or SAS business analytics software suite, suggests that things are not settled. Indeed the lively growth of R and the fact that more data miners say they use R than any other software (including Excel spreadsheets) suggest that data mining is a very unstable operation. Hence, the question of how R is invoked and works to create code around Facebook data (or any other) is a live one. It is invoked statistically, and statistically here means in relation to large numbers. The problem that Facebook analysts deal with is how to make sense of the scale of their own growth. They almost have to treat their own platform statistically, as a population of users, whose activities in space and time need to be modelled in order to derive statistics that can be used to make inferences about them. What does it mean to to write statistical code for accumulated data? The association between computation and accumulated population data is not new. We need only think of Hollerith machines and the use of computation to produce statistical tables for population censuses.[23]

The code that Itamar Rosenn and the Facebook analytics team wrote draws on the R package, *lars. Lars,* standing for least angle regression, was written by statistical machine learning researchers at Stanford, and in particular, by a leading figure in the field of machine learning, Trevor Hastie.[24] The relevant scientific publications date from 2004.[25] The package itself, *lars*, went into CRAN, the R equivalent to Perls cpan, in May 2007. One or two years later, Facebook used it to model user responsiveness, and to predict who will return to Facebook and who won't. The shift from scientific publication by academic statisticians to code in R and then to code at Facebook was rapid. It only took five years. Unlike Varian and Choi, the Facebook team have not published their results as a research paper. They are rather more pragmatic in their approach to R. Rather than use it to shape a discourse on the value of data, they make use of it to give form to the data skimmed off the incessant accretions of Facebook. To explore the dynamics of Facebook's growth, Rosenn looked more closely at the intersection between registration and other desiring flows traversing the software. They found, for instance, that the probability of someone returning to Facebook after registering for an account has some correlation with the ways in which they configure their Facebook page. Any degree of customization is correlated with increased retention. Hence, anything that invites or affords customization could be a useful accumulation strategy. Again, like Varian's, these results are hardly surprising.

Perhaps the actual results are less important than the ways in which they were achieved. The *lars* approach is used as a way of selecting from many possible models of the data at hand. As the authors of the original *lars* publication write,

> typically we have available a large collection of possible covariates from which we hope to select a parsimonious set for the efficient prediction of a response variable. *Lars*, a new model selection algorithm, is a useful and less greedy version of traditional forward selection methods.[26]

In other words, the *lars* technique that the analytics team has picked up from R allows them to try many different linear regression models and efficiently choose a predictive one. The 'response variable' they seek – whether someone will return to Facebook after registering – is estimated not by simply building a model that predicts, but by trying many models of prediction in a carefully regulated order. The techniques of linear regression are extremely well established, but every linear model requires individual design and interpretation. The statistical modelling of Facebook users will remain rather flat-footed unless there are ways to try many different models. Yet building numerous models is a laborious and uncertain undertaking. Here the packaging of recent statistical research into R enables 'less greedy' (that is, less likely to settle on a locally optimal) treatment of data. Greedy algorithms, while simpler to design and implement, often do not find globally optimum solutions.[27] They search locally for the nearest best solution to a problem. *Lars* offers a way of automatically building and trying many different models with less chance of being trapped in some sub-optimal result. In this case, a sub-optimal result would be to select a model of Facebook user behaviour that did not include relevant predictors, and that therefore missed possible lines of axiomatization that Facebook analysts could develop as part of their accumulation strategy.

Predicting what R programmers will do

R programming explores how people or things act (what they do on Facebook, how likely they are to book a flight to Hong Kong, what car they will buy, etc.). As we have just seen, the data structures that R brings to bear on events, actions, behaviours, beliefs and desires are statistically informed models that transform tables of data into correlations, clusters, imputations, inferences, confidence intervals and scores of various kinds. These transformations in some ways seek to control data, but in other ways, they make that data more fluid, labile and mutable. The tension here between aggressive control over data and the potential for free association through data is quite marked. While methods and techniques promise control, the proliferation of techniques and methods risks shattering the very possibility of control. In a sense, the very reason programmers get excited about R – it makes available so many methods that promise to corral the data – also potentially undermines their investment in it.

It should come as no surprise then that the practice of R programming itself should become a target of prediction. Take a recent Kaggle competition,[28] sponsored by the data blog Dataist.com. Kaggle sponsors data-mining or statistical analysis competitions ranging across health data, chess ratings, tourism, grant application outcomes, dark matter and the progression of HIV infections. In the case of the 'R Package Recommendation Engine' competition, competitors were supplied with meta-data from the CRAN repository describing which of 1,852 R packages were installed by 52 R users. This is the so-called 'training data' used to build a predictive model. Having modelled this sample population, competitors were judged on how well they could predict what further packages were installed by much larger group of R users, the test group. A good 'recommendation engine' would provide new users of R with suggestions on as to which packages they should install. The training data-set for the competition comes directly from CRAN, so it itself offers an interesting view on how data is being worked on globally. A time-series analysis of package installation in R would make it possible to see the distribution and mobility of data methods in flight. I am less interested here in what is being done with R packages than in the

Kaggle competition on data in R. The sample solution provided by the competition organizers suggests something about the forms of thought involved:

```
library('ProjectTemplate')
try(load.project())
logit.fit <- glm(Installed ~ LogDependencyCount +
                 LogSuggestionCount +
                 LogImportCount +
                 LogViewsIncluding +
                 LogPackagesMaintaining +
                 CorePackage +
                 RecommendedPackage,
    data = training.data,
    family = binomial(link = 'logit'))
summary(logit.fit)
```

The statistical model at the core of the sample solution is very simple: 'logistic regression', a form of linear modelling that allows the probability of an event to be predicted on the basis of several variables whose relative contributions to the event are not known.[29]

The technique is widely used in marketing research. Here the event to be modelled is whether a particular R package has been installed or not. The variables are all the dependencies, maintenance status, existing recommendations and so forth. The sample model yields a set of probabilities for every R package for every R user in the test data set. The competitors in the Kaggle competition try to improve the sample solution. Competitors can make as many entries as they want, but after the judging deadline, the solution with the highest predictive power wins. Attentive readers may have noted that the proposed sample solution bears a great deal of similarity to the Facebook user retention analysis. Indeed, Facebook's model also used logistic regression, but coupled it with a technique for modelling models (*lars*). Rather than one model, they built many. But even this difference begins to disappear if we think of the Kaggle competition as about the modelling of modelling. Rather than using a statistical technique of *lars* to check many models, it uses the competition format to generate and check many models. Furthermore, these models are models of how data is being modelled, and how the modelling of data is being embodied by programmers using R, by in other words, R users.

There is a striking symmetry between the situation faced by Facebook analyists in trying to understand how people become habitual users, and the

Kaggle competition in trying to predict how programmers who become users of R use R. Here we might see most directly how users become useRs! (with all the factorial and imperative senses of that term). Once the embodiment of data structures and the desires to munge data are also processed in a predictive analytics recommendation engine, unruly bodies of R code can be brought into the hallucinations of data productivity. This is both satisfying and exciting. The satisfaction comes from the fact that useRs become more like users with calculable response rates and retention ratios. The excitement comes because the attempt to model useR!s exacerbates the mobility of methods flowing through R.

Pure love of data

What would be a good embodiment of data structures? To embody a pure love of data, if such a love were possible, would be tantamount to disinvestment. Disinvestment is a very difficult position to maintain amidst the economic-affective flows of productivity. Indeed, desiring investment in programming is the precondition for alterity in software, and, in relation to data. The underlying problem here, as in so many forms of investment, is to know how the excitement of satisfied aggression shifts into a non-destructive eroticizing of data. R programmers would no doubt find this an odd statement. The point, however, is that only through the associative processes given off by unbound affects can un-analytic un-predictive events take place. Hence eroticizing data is a way of talking about the need for hitherto unexpected forms of exchange and encounter.

There are indications that something complicated is happening to data. Desires to do things with data are shifting. Although there is much data talk today about the accumulation of data, the accumulation of data is not the relevant point of entry. We must look in other places. Interference and intersection of the lines of desire that traverse individuals are much better places to learn about the points of inflexion and free association. The body of R code and the bodies of R programmers are interesting places because they take populations as aggregates and multiplicities to heart, or more precisely, into their own hands. R is an encounter with the vitality of populations. What does it mean for software to encounter populations in this way? In *Anti-Oedipus* Deleuze and Guattari wrote: 'in the unconscious, there are only populations, groups and machines'.[30] That statement, so well known and widely invoked, is hardly ever read in terms of the practices and epistemics of populations – statistics, demography, epidemiology and assorted field sciences. We see the statistically mediated encounter between coding and populations taking place in contemporary data-driven analytics such as Google's seasonal autoregression, Facebook's *lars* or Kaggle's logistic regression, suggesting that populations, groups and machines are being brought together. Populations can be coded or undo coding since their character is recombinant, chance-laden and associative. The question is to what extent bodies of code themselves can become population aggregates and micro-multiplicities. This is not only because all coding techniques are partial objects subject to re-connection and re-configuration, but also because so much coding, as abundantly attested by the population of R packages, scripts, graphics and derivate mediations, has no particular productive aim and indeed constitutes a break or rupture in the productive flows. 'It is only desire that lives from having no aim'.[31]

Notes

1 Bersani, Leo and Phillips, Adam, *Intimacies* (Chicago: University of Chicago Press, 2008), 66.

2 Brennan, T., *Exhausting Modernity: Grounds for a New Economy* (London and New York: Routledge, 2000), 29.

3 Bersani and Phillips, *Intimacies*, 67.

4 Ibid., 72.

5 Phillips, Adam, *Equals* (New York: Basic Books, 2003), 21.

6 R Development Core Team, 'The R Project for Statistical Computing' (2010), http://www.r-project.org/ (accessed 12.12.2013).

7 Muenchen, Robert, 'Popularity – r4stats.com' (2011), http://sites.google.com/site/r4statistics/popularity (accessed 24.05.2011).

8 CRAN, 'The Comprehensive R Archive Network' (2010), http://www.stats.bris.ac.uk/R/ (accessed 5.05.2010).

9 Hacking, Ian, *The Taming of Chance* (Cambridge: Cambridge University Press, 1999).

10 Desrosières, Alain, *The Politics of Large Numbers: A History of Statistical Reasoning* (Cambridge, MA: Harvard University Press, 1998).

11 Prediction Impact Inc., 'Predictive Analytics World Conference: Agenda, 2009', http://www.predictiveanalyticsworld.com/sanfrancisco/2009/agenda.php#usergroup (accessed 12.12.2013).

12 Hardy, Q., 'Power in the Numbers', *Forbes.com* (2010), http://www.forbes.com/forbes/2010/0524/opinions-software-norman-nie-spss-ideas-opinions_2.html (accessed 12.12.2013).
13 Campbell-Kelly, Martin, *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry* (Cambridge, MA: MIT Press, 2003).
14 R Foundation for Statistical Computing, 'UseR! 2011' (2011), http://www.warwick.ac.uk/statsdept/user-2011/ (accessed 12.12.2013).
15 Deleuze, Gilles and Guattari, Félix, *Anti-Oedipus: Capitalism and Schizophrenia*, trans. Robert Hurley, Mark Seem and Helen R. Lane (New York: Continuum, 2004), 348.
16 Ibid., 322.
17 Cramer, Florian, *Words Made Flesh: Code, Culture, Imagination* (Rotterdam: Piet Zwart Institute, 2005).
18 Manovich, Lev, *The Language of New Media* (Cambridge, MA: MIT Press, 2001).
19 Hastie, T., Tibshirani, R. and Friedman, J. H., 'The Elements of Statistical Learning: Data Mining, Inference, and Prediction' [シュプリンガー・ジャパン株式会社] (2009).
20 Raymond, E., 'munging', *The Jargon File* (2003), http://www.catb.org/jargon/html/B/brick.html (accessed 12.12.2013).
21 McKinsey & Company, 'Hal Varian on how the Web Challenges Managers', *McKinsey Quarterly – Strategy – Innovation* (2009), http://www.mckinseyquarterly.com/Hal_Varian_on_how_the_Web_challenges_managers_2286 (accessed 12.12.2013).
22 Varian, H. and Choi, H., 'Official Google Research Blog: Predicting the Present with Google Trends' (2009), http://googleresearch.blogspot.com/2009/04/predicting-present-with-google-trends.html (accessed 12.12.2013).
23 Beniger, James R., *The Control Revolution: Technological and Economic Origins of the Information Society* (Cambridge, MA: Harvard University Press, 1997).
24 Hastie, T. et al. 'The Elements of Statistical Learning': data mining, inference, and prediction [シュプリンガー・ジャパン株式会社] (2009).
25 Efron, B. et al., 'Least Angle Regression', *The Annals of Statistics*, 32(2) (2004): 407–51.
26 Ibid.
27 Black, P.E., 'greedy algorithm', *Dictionary of Algorithms and Data Structures* (2005), http://xlinux.nist.gov/dads//HTML/greedyalgo.html (accessed 12.12.2013).
28 Kaggle Pty Ltd, 'Description – R Package Recommendation Engine – Kaggle' (2011), http://www.kaggle.com/c/R (accessed 12.12.2013).
29 Cramer, J. S., 'The Early Origins of the Logit Model', *Studies in History and*

Philosophy of Science Part C: Studies in History and Philosophy of Biological and Biomedical Sciences, 35(4) (2004): 613–26.

30 Deleuze and Guattari, *Anti-Oedipus*, 283.

31 Ibid., 367.

6

Do (not) Repeat Yourself

Michael Murtaugh

Duplication (inadvertent or purposeful duplication) can lead to maintenance nightmares, poor factoring, and logical contradictions.

Duplication, and the strong possibility of eventual contradiction, can arise anywhere: in architecture, requirements, code or documentation. The effects can range from mis-implemented code and developer confusion to complete system failure.

The Mars Climate Orbiter was lost due to a semantic contradiction: part of the system was working in Imperial units, another in Metric. There was a duplication of knowledge (implicit units), and the duplicates were out of step.

The Problem:

But what exactly counts as duplication? CloneAndModifyProgramming is generally cited as the chief culprit (see OnceAndOnlyOnce, etc.), but there is more to it than that. Whether in code, architecture, requirements, documents or user documentation, duplication of knowledge – not just text – is the real culprit.

Therefore:

The DRY (Don't Repeat Yourself) Principle states:

Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.

[...]

It's a battle of two really strong urges – OnceAndOnlyOnce vs avoiding PrematureGeneralization. Do I duplicate for now and try to live with the duplication for a while, or violate YagNi (YouArentGonnaNeedIt) and come up with some half-cocked generalized solution? It's a tough one, because almost all programmers hate duplication; it's a sort of primordial programming urge.[1]

This long quote is the opening section of the page titled 'Don't Repeat Yourself' (DRY) on the Portland Pattern Repository (PPR). The pages of the website, devoted in part to the practice of programming, are heavily cross-linked, forming a messy tangle of programming paradigms and self-help tips for improving code with CamelCased hyperlinks,[2] often in the form of an imperative such as RefactorMercilessly, PutThingsWhereYouLook and SeparateTheWhatFromTheHow. At the same time it contains seemingly contradictory meta-tips that warn against SilverBullet and OneTrickPony solutions. Despite the apparent clarity and appeal for 'authoritativeness', later in the same page, the discussion presented above takes on a slightly different tone:

> But: It may take time and effort to select and/or create a definitive source – see YouArentGonnaNeedIt. – DickBotting
>
> Hmm. 'Every piece of knowledge must have a single, unambiguous, authoritative representation within a system'. Is this not a reasonable definition for a Singleton?
>
> No. DRY refers to your source code, not your running program. ThirdNormalForm is the analogous principle for data.
>
> Ugh. I think you're both barking up the wrong trees there. It doesn't refer to source code, nor to the running program. It refers to a system. Needn't have computers involved at all.[3]

The PPR is the product of a particular community of programmers. It is in fact the 'mother of all wikis', and was created (and maintained) by Ward Cunningham, who coined the term 'wiki'. The fact that the discourse of the PPR takes place on, and came about in conjunction with the development of, the wiki, is significant. While superficially a wiki page provides for a kind of 'single source', its great strength lies in the fact that by containing the entire history of edits, by permitting editing by anyone and by allowing differences of opinion to be made explicit, the wiki itself is far from being free of contradiction or duplication.

The language of seeking 'single, authoritative, unambiguous' knowledge by the architects of the software that would go on to inspire the more popular Wikipedia project seems to sadly devalue its own core strengths. While the prescriptive nature of the DRY discourse appeals to a programmer's sense of utility and efficiency, it also seems to be in denial of the very nature of the practice it aims to serve.

Programmers walk a fine line between seeking ecstatic singularities while at the same time enduring dutiful and crushing conformity to correctness and convention, performing a practice that is by nature highly repetitive. Programmers often have to search for a balance between considering the details of a particular situation and flying into euphoric quests for abstraction, desiring

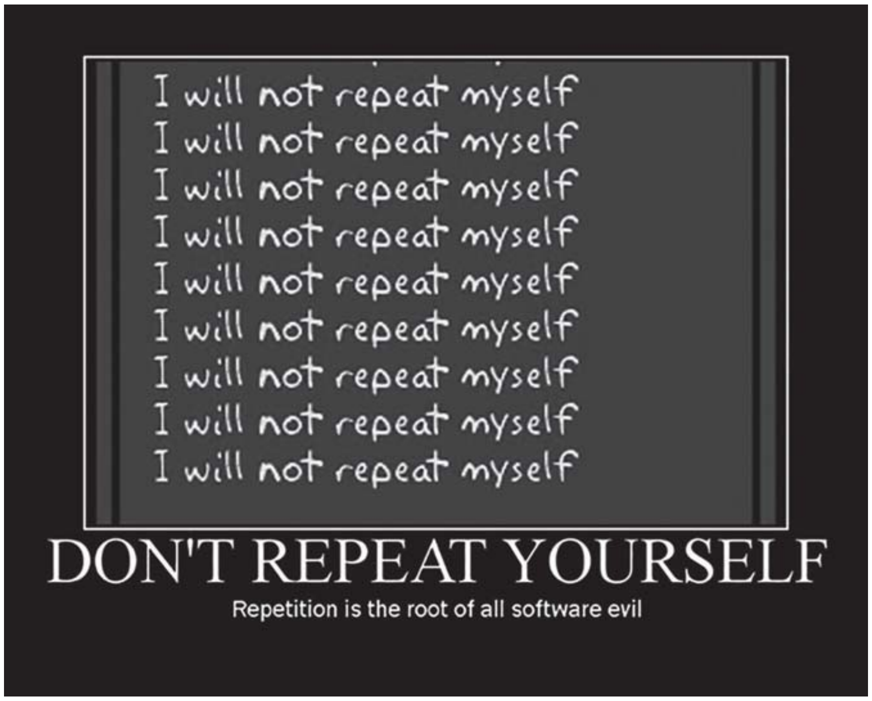


Figure 6.1 Don't Repeat Yourself
Source: Unknown.

code that goes beyond the satisfaction of some particular need into the ecstatic realm of the unforeseen and unexpected. The quasi-ejaculatory nature of this process is evident in another of the PPR principles, that of avoiding 'premature generalization'.

Popular web programming frameworks, such as Ruby on Rails and Django, pride themselves on adhering to the 'principle of DRY'. The Rails framework initially promoted itself as a response to the drudgery of website programming and was unique in listing the 'joy' of using it, among its other technical features, as a way of winning over programmers: 'Rails is a full-stack, open-source web framework in Ruby for writing real-world applications with joy and less code than most frameworks spend doing XML sit-ups'.[4] Such 'joy' is contrasted with repetition, something much less desirable and seen as valueless, with its essential qualities remaining unrecognized.

However, it is often the programmers' recognition of a pattern already learned through repetition that is most compelling in the use of a particular framework. In the same way that one feels lost and isolated in the woods and then reassured when one comes across a well worn path, frameworks are condensations of practice and reinforce a sense of community among their practitioners. In contrast to the practice of belonging, the fun to be had in the pioneering discovery of the novel or unique is isolating. In addition to this, repetition is intrinsically intertwined with the development of the craft of programming.

> [...Skill] development depends on how repetition is organised. This is why in music, as in sports, the length of a practice session must be carefully judged: the number of times one repeats a piece can be no more than an individual's attention span at a given stage. As skill expands, the capacity to sustain repetition increases. In music this is the so-called Isaac Stern rule, the great violinist declaring that the better your technique, the longer you can rehearse without becoming bored. There are 'Eureka moments' that turn the lock in a practice that has jammed, but they are embedded in routine.[5]

There can be a tangible pleasure in quickly typing out the template of a familiar programming structure. Far from celebrating the birth of a unique new creation from scratch, it is rather a joyful expression of the pattern that increasingly becomes physically embodied in the programmer him/herself. Here, the material that one once struggled with, with time becomes something ingrained in 'one's fingers'.

On the surface, the black box of abstraction promises the programmer that if he/she can only get the abstraction right, he/she will never have to deal with a particular kind of problem again. In fact, working with abstraction is a gradual process, inclusive of struggling repeatedly with the material of a problem and, thus, acquiring the skill that would feel 'natural and easy' by becoming a part of the body of the programmer. Rather than removing the problem, repetition produces increased capacity to deal with the problem and, thus, the problem can be repeatedly successfully tackled. The formalization of abstraction in the form of code (the syntax and naming of a function object, for instance) can be seen as merely a culmination of the necessary prerequisite of practising a repetitive process that made its recognition and recall possible.

Working with recursion is a particular kind of programming skill that often takes a great deal of practice before it is fully mastered. Once it is learned, there is a self-effacing wonder in watching 20 lines of code dissolving down to ten, then five, as all the 'edge cases' that could possibly be imagined map onto the few folds of a particular recursive structure. Similar to the pleasure of kneading dough, working with recursion is about the almost miraculous transformation of code through repeatedly working it.

Repetition is an essential part of the process of recognizing and constructing abstractions. The fact that experienced programmers might directly write code using concise and 'correct' abstractions is more a reflection of their experience than an absolute (and transferable) measure of quality.

Code smells

> If you are aware of CodeSmells, and duplicate code is one of the strongest, and you react accordingly, your systems will get simpler. When I began working in this style, I had to give up the idea that I had the perfect vision of the system to which the system had to conform. Instead, I had to accept that I was only the vehicle for the system expressing its own desire for simplicity. My vision could shape initial direction, and my attention to the desires of the code could affect how quickly and how well the system found its desired shape, but the system is riding me much more than I am riding the system.[6]

My great Aunt Margaret, a piano teacher and former Catholic school principal, would implore all overnighting nieces and nephews to take a shower in the morning, to counter the 'beddy smell'. Naive as I was, it was only years later that it occurred to me how the term 'beddy' was in fact a thinly veiled euphemism for the less comfortable subject of one's 'body'. Just as my Aunt displaced the body by the bed, the struggles of programming often get projected from programmers onto the code. Bad code has a smell, independent desires and an ability to ride or be ridden by the programmer. Repetition can be experienced as a tiring physical exercise.

In addition to devaluing repetition as something smelling 'bad', something to be either absolutely avoided or at best tolerated, the transference of a smell to code is also indicative of another set of displacements, such as the code being separated from the practice of programming, and the practice of programming being separated from the physical effort required of the body of the programmer him/herself. Coding can be, and often is, physically exhausting work, as illustrated by the passage below:

> Just got off the phone with F. Feeling slightly remorseful at being kind of pissy and short. Maybe need to write an email: Sorry I was vague. Orality impaired – I could better write an email. I've been coding intensely over the past 4 days. I was

> only half-listening as F ran down the planning for the workshop – discounting that information that I already knew (annoyed at the redundancies)... a Borg voice speaks to me from the collective: this information is not relevant; this conversation is inefficient. I'm having difficulty following what's being said.
>
> Thinking of the 'code smells' – after these long stretches of coding the smells take a physical form – though it's not coming from the code … I'm unable to tell if the unpleasant odors I seem sporadically aware of are originating from rotting garbage in the bin or from me. Reminiscent of baby diapers. Probably the garbage bag … must be.
>
> Dim the screen – too bright. On with coding...
>
> I can no longer come up with meaningful names for things. Have started using names like 'aa', 'bb', and 'aaa'. Switching between these abstract symbols seems easier; reduced semantic overload = less need to think.
>
> Can no longer remember what task I am currently working on. I start writing down tasks not 'to do', but what I'm supposedly 'doing now', so that each time I slip, I can refer to the note. Need to update the model and regenerate the database before going to bed. Before I forget why it's important.[7]

For programmer Will Crowthers, commenting on his hobby below, rock climbing does not merely provide an escape from programming:

> You would have to forget about everything. When you're rock climbing, you must not think about anything but the rock climbing or you're apt to get killed. And it just wipes everything out for a day or two or whatever it is. However long you're off climbing, which tended to be a weekend, you don't think of much else.[8]

Crowthers' 'extreme hobby' parallels the intensity of his experience of programming itself. He needs an escape with a sufficiently matching intensity to give him a break from that of the engrossingly 'disembodied' practice of coding. Crowthers is best known for programming the early computer game classic 'Collossal Cave Adventure' (also known as 'Adventure') in his spare time while working at the company BBN Technologies in Massachusetts in the late 1970s. A pioneering example of an interactive program that simulates the experience of cave exploration, the writing of the game was, according to Crowthers, in part an attempt to reconnect to his children after an estrangement from his wife whom he had met while caving.[9]

'Extreme Programming', a concept whose origin and development can be traced to Kent Beck on the PPR,[10] has become an important movement concerning rethinking traditional approaches to software programming practice. Beck's choice of the term 'extreme programming' clearly invokes 'extreme sport', and indirectly references the often frustrated desires of a programmer to experience, in their practice of programming, the intensity of physical experience such as that described by Crowthers.

According to the principles of DRY, it would seem that the job of a programmer is to detect patterns and to fold these into redundancy-free perfection. This suggests an ideal, Plato-inspired practice of programming, wherein the programmer, after meditative moments of reflection, is able to effortlessly condense the chaotic cacophony of the reality around him/herself into a stream of precise expressions, gliding from unique formulation to unique formulation and never looking back.

> People imagine that computer programming is logical, a process similar to the one of fixing a clock. Nothing could be further from the truth. Programming is more akin to an illness, a fever, an obsession. It is like riding a train and never being able to get off.[11]

Programmer and journalist Ellen Ullman compares software design to using methamphetamine, as the 'speed high is the only state that approximates the feel of a project at its inception. Yes, I understand. Yes, it can be done. Yes, how straightforward. Oh yes, I see'. The trip is, however, brought to an abrupt end when 'you write some code, and suddenly there are dark, unspecified areas'.[12]

Ullman's description of how the transition from the plan to the writing of code drops from the luminous clarity of a pre-implementation specification into the dark areas of the unspecified seems to invoke something of the fear and pleasure of Crowther's cave explorations. The experience of working with code can be an exhilarating modulation between the light and the dark, between losing and regaining one's footing, between the logical and the absurd.

Do repeat yourself

> There is always a reason for missing an easy toss. Repeat toss and you will find it. If you rap your knuckles against a window jamb or door, if you brush your leg against a desk or a bed, if you catch your feet in the curled-up corner of a rug, or strike a toe against a desk or chair go back and repeat the sequence. You will be surprised to find how far off course you were to hit that window jamb that door that chair. Get back on course and do it again. How can you pilot a spacecraft

> if you can't find your way around you own apartment? It's just like retaking a movie shot until you get it right. And you will begin to feel yourself in a film moving with ease and speed. But don't try for speed at first.[13]

In the short story 'The Discipline of DE', William Burroughs, in the guise of a retired Colonel Sutton-Smith, describes to the reader the joys of living a life according to the prescripts of 'Do Easy'. The text is written primarily in the second person: a parodic fusion of self-help guide and military pep talk. While DE's 'do repeat yourself' paradigm would ostensibly seem to oppose the 'don't' of DontRepeatYourself, the two have a lot in common. When Burroughs writes: 'once you find the easy way you don't have to think about it, ...it will almost do itself',[14] one can hear echoes of Kent Beck's 'mystical' musings of a systems' 'own desire for simplicity', cited above. DE's message of 'repeat until perfection' captures much of the reality of software design practice: a frequently obsessive attention to detail and process, a tendency towards excessive (self-) optimization and an aesthetization of efficiency.[15] The promises of ease and speed could be taken straight from the copy of a new programming framework.

> Everyday tasks become painful and boring because you think of them as WORK something solid and heavy to be fumbled and stumbled over. Overcome this block and you will find that DE can be applied to anything you do even to the final discipline of doing nothing. The easier you do it the less you have to do. He who has learned to do nothing with his whole mind and body will have everything done for him.[16]

Burroughs' reductio ad absurdum reveals a dark side to DRY in its relation to a larger software industry. In a system where code is a product to be protected and exploited commercially, the efficiency of the process tends to eliminate the usefulness of the programmer; truly efficient coding would lead to a point where the coder him/herself becomes 'redundant', expendable. Burroughs' final image of the mastery of doing nothing leading to a position of privilege and power concisely reveals the implicit motivations behind much of the venture capital interest in software development.

The GNU project was the response of one programmer, Richard Stallman, to what he felt were the injustices of a software industry that separated the programmer from the product of his/her labour through nondisclosure agreements and restrictive software licenses. The GNU project and the ensuing Free Software movement, encourage a practice of software development whereby code is released under a license that ensures that it remains not only freely usable, but also reworkable and redistributable by subsequent programmers. In 'freeing' the code, the General Public License (GPL) shifts value from the code to the surrounding practice. The value of free software is the community of developers, documenters, researchers, designers and users which is rather than the 'shrink-wrapped product' or 'killer app' *per se*.

Even with free software's fundamental shift in value from code to community, Stallman's early manifesto still includes a 'DRY' stance as one of the core 'benefits' of the project. '[The GNU project] means that much wasteful duplication of system programming effort will be avoided. This effort can go instead into advancing the state of the art.'[17] As with the PPR, arguments for efficiency seem to be inevitably made even when they contradict the realities of the practice.

The free software community is a rich tapestry of duplication, forked projects and reinventions of the proverbial wheel. The term 'yet another' is common in the names of free software projects as a humorous way of acknowledging (and gently atoning for) the redundancy.[18] Recursion and contradiction play a substantial role in programmer humour. The GNU name itself (standing for 'GNU's not UNIX') is a kind of nerd joke, doubly contradictory both as a version of UNIX that *is not UNIX*, and inherently incomplete in its recursive definition. In a similar way the very formulation of 'Don't Repeat Yourself' as a kind of a programmer's mantra, and thus to be recursively repeated, is also absurd.

The negative implications of separating code from practice are many: formal software instruction is pervasively discouraging to beginners and the 'uninitiated'.[19] The labour of software design is easily exploitable and software professions are precarious, whereas the economic forces promote the fragile and decontextualized product of code and ignore its larger sustaining community.

As a programmer, I 'get' DRY and I value Beck and the PPR in their contributions to software and to the discussion of software practice. The problem is that maxims, such as 'Don't repeat yourself' only work when they are not taken literally, and when their implicit values are questioned. There is a continual need to (re)value software practice and avoid reducing it to a kind of 'shortest path' problem.

Software practice includes logical contradiction, necessitates 'bad' code and requires repetition. When, in the definition of DRY, the 'duplication of knowledge, not just text' is defined as a core part of 'the problem' to be solved, it exposes an impoverished conception of knowledge isolated from practice.

In software design, as in other forms of cultural discourse, redundancy and repetition are essential to the necessarily incomplete processes of knowledge production, practice, circulation and maintenance.

Notes

1 See the page 'Don't Repeat Yourself', http://c2.com/cgi/wiki?DontRepeatYourself (accessed 12.12.2013), or, for an archived snapshot version circa 2005, see http://web.archive.org/web/20050403083926/http://c2.com/cgi/wiki?DontRepeatYourself (accessed 13.06.2013).

2 CamelCased hyperlinks are links in which capitalized words are merged into compound words, sometimes used in Wiki markup languages for key terms that become automatically linked to other Wiki pages. See the entry on Wikipedia: http://en.wikipedia.org/wiki/CamelCase (accessed 13.06.2013).

3 See the page 'Don't Repeat Yourself'.

4 The 'Ruby on Rails' website soon after its launch in 2005 – archived link, http://web.archive.org/web/20050601015146/http://www.rubyonrails.org/ (accessed 13.06.2013).

5 Sennett, Richard, *The Craftsman* (New York: Penguin, 2008), 38.

6 See the page 'Once and Only Once', http://c2.com/cgi/wiki?OnceAndOnlyOnce, citation signed 'KentBeck, feeling mystical, see MysticalProgramming' – another page at PPR (accessed 13.06.2013).

7 Personal notes written during programming work.

8 Crowthers, Will, scanned transcript of an Advanced Research Project Agency (ARPA) interview, page 4, downloadable from http://www.archive.org/details/WillCrowtherInterview (accessed 13.06.2013).

9 Ibid., 1, 4.

10 See http://c2.com/cgi/wiki?ExtremeProgramming (accessed 13.06.2013).

11 Ullman, Ellen, 'Out of Time: Reflections on the Programming Life', *Educom Review*, 31(4) (1996).

12 Ullman, Ellen, *Close to the Machine: Technophilia and Its Discontents* (San Francisco: City Light Books, 1997), 21.

13 Burroughs, William, 'Word Virus', in *The William Burroughs Reader* (London: Flamingo, HarperCollinsPublishers, 1999), 386–92.

14 Ibid., 390.

15 See Fuller, Matthew, 'Elegance', in *Software Studies*: *A Lexicon*, ed. Matthew Fuller (Cambridge, MA: MIT Press, 2008), 87–92.

16 Burroughs, 'Word Virus', 391.

17 GNU Manifesto (1985), http://www.gnu.org/gnu/manifesto.html (accessed 13.06.2013).

18 See the 'Yet Another' page, http://en.wikipedia.org/wiki/Yet_another (accessed 13.06.2013).

19 See, for instance, Margolis, Jane and Fisher, Allan *Unlocking the Clubhouse: Women in Computing* (Cambridge, MA: MIT Press, 2003). A quote from Papert can also be useful here: 'Children do not follow a learning path that goes from one "true position" to another, more advanced "true position". Their natural learning paths include "false theories" that teach as much about theory building as true ones. But in school false theories are no longer tolerated. [...] Our educational system rejects the "false theories" of children, thereby rejecting the way children really learn'. Papert, Seymour, *Mindstorms: Children, Computers, and Powerful Ideas* (New York: Basic Books, 1993), 132.

7

Not Just for Fun

Geoff Cox and Alex McLean

There is something inherently human about the ability to perform creative actions with verbal language expressed in the form of jokes. To Paolo Virno, jokes are not simply Freudian clues to the unconscious, but diagrams of innovative action that represent how humans can diverge from social norms.[1] This chapter argues for something similar in the way fun is had with software which exemplifies the innovative action of code in addressing software norms. This indeed is common in critical approaches to making software, where fun often indicates the use of irony or satiric humour by programmers to jolt the user out of their usual interactions with normative tools and productive work. But, as the distinction between work and play becomes ever more blurred, fun is an expected part of all kinds of serious production; indeed, like most of lived experience fun has become subsumed into the 'social factory'.

That fun can be had with software is demonstrated in the numerous titles that make explicit reference to it: for instance *Fun with Computers* by T. Ramasami, or the many examples of technical manuals that try to convince the user that things are far easier and more fun than the otherwise serious reality of coding. A quick search reveals numerous examples of this tendency: *Fun and Games with the Computer* by Edwin R. Sage; *Fun with Computers* by A. Roy Chowdhury; *Fun with Computers and BASIC* by Donald D. Spencer; *Fun with Computer Electronics* by Luann Colombo and Peter Georgeson; (and even more assertively) *Computers are Fun* by Tony Gray and Carl Billson; and *The Fun of Programming* by Jeremy Gibbons and Oege de Moor.[2] Fun is a useful way of encouraging productive work with computers and viewing some of the underlying difficulties as enjoyable challenges. This is perhaps particularly noticeable with game development, exemplified by Raph Koster's *A Theory of Fun for Game Design*, where the inherent fun of computer games is understood through cognitive science as 'the feedback the brain gives us when we are observing patterns for learning purposes', taking players beyond mere entertainment value and suggesting 'alternatives to fun'.[3] Deviating from the norm in this way is widely regarded as an essential part of the process of capitalist innovation as well as its critique; fun in this sense seems to have become instrumentalized.

A further example is Linus Torvalds's book title *Just for Fun: The Story of an Accidental Revolutionary*, written with David Diamond, which is part autobiography and at the same time charts the serious development of the GNU/Linux operating system.[4] The development of the Linux kernel makes a good case study as it emerges from a culture of sharing across code repositories and software distribution networks, like sharing a good joke. Yet this is to be expected as the history of sharing is as old as the sharing of recipes, as Richard Stallman puts it, 'as old as computers, just as the sharing of recipes is as old as cooking'.[5] With free software development, each individual's work is valued in the context of the multiple efforts of all contributors, indeed it emphasizes that innovation and value creation need to account for a deeper understanding of collaborative processes and the social relations that arise from these cultures of sharing.

This point is also what Adrian Mackenzie develops when he claims that the Linux kernel represents a particularly unstable relation to commodified software and hardware as performative action. He thinks that the way in which Linux is produced and continually changed cannot be separated from its performative structure as code, and describes it as a performative 'speech act' that produces an uncertain relation between the code object (the Linux kernel) and the code subject (the programmers who use it), and thus challenges its property relations and corporate relations of production.[6] It diverges from the software norm and innovates in ways that follow the logic of free software development. Fun in this case coexists with the serious business of making source code freely available and open to further modifications. But rather than declaring development by accident in the case of Torvalds, this chapter makes reference to Virno's conception of jokes as an expression of innovative actions drawn from the kernel of political possibilities across human and program languages. Thus the chapter is a speculation on the source code of (software) jokes, and explores this at the intersection of human and machine interpretation.

On jokes

The point for Virno is not the content of jokes, but that newly invented forms diverge from established rules and perceived norms as a result of the innate creativity embedded in language itself. The chapter extends this to programming language, to coding practices and, in particular, the ways that code jokes most often do not operate within the confines of formal language, but either outside it in comments, variable names and layout (as with code poetry or codeworks) or in the development of playful esoteric languages (such as brainfuck and befunge, which will be introduced later). It is not that jokes are embedded in the machine's unconscious, of course, but that jokes can be based on innovative deviations from the conventions of coding practices and their formal expression and interpretation.

Before discussing Virno's ideas in more depth, it is worth considering some other commentaries on jokes and the phenomenon of humour from the wider sources at hand. There are far too many to mention but Sigmund Freud's reflections on jokes and the 'wittiness' of dreams is clearly an important reference. In *The Joke and Its Relation to the Unconscious*, first written in 1905, Freud draws upon his theory of the unconscious to explore the idea that jokes express hidden desires and fantasies.[7] Joke-work, like dream-work, is thereby understood as a means of negotiating the ways that consciousness censors and disguises intellectual processes. Joke techniques of condensation, displacement, representation by absurdity or by the opposite, indirect representation, and so on are all present in dreams. But whereas the dream is asocial and serves to spare displeasure, jokes are distinctly social and operate to gain pleasure by releasing inhibitions and discharging excess energy through laughter which involves the whole body. Both share the complex workings of the psyche that does not express itself by accident even when appearing to mash up ideas or develop an operating system. This social aspect of jokes is crucial.

Writing more recently, the philosopher Simon Critchley has examined the importance of humour, emphasizing how a change of situation exerts a powerful critical function, of particular relevance to creative practices. In an interview for *Cabinet* magazine in 2005, drawing upon his book *On Humour* (2002), he emphasizes humour as a social practice that reveals wider social insights.[8] Repression is clearly part of this, with the example given from Plato's *Republic* where it is explained that the guardians of the *polis* were not meant to laugh because laughter was considered too uncivilized and bestial. It is because laughter is considered deviant that jokes are seen to exhibit the potential to be a powerful critical tool.

Yet Critchley also explains how witticism (as a sophisticated style of humour) is also considered to be a sign of advanced civilization as opposed to common jokes, where witticism is tied to the development of liberal democracy, and characterized by the figure of the dandy or wit. The witticisms of Slavoj Žižek seem to exemplify this, including his many jokes related to the subject of really existing Socialism.[9] One example he gives is a joke that reveals the futility of dissident protest and the lack of recognition of wider conditions. His claim is that no one really took the subject of totalitarianism seriously but rather as a joke; and that progressive intellectuals who thought it was serious merely indicated the futility of their protests. As we can see from the above, the function of humour for Žižek lies in its ability to invert common sense, following the Marxist-Hegelian logic of dialectical reversal. Moreover, he conceptualizes such things in terms of the 'return of the real' (reworking Freud's 'return of the repressed'), where impulses previously repressed erupt into social life at unexpected moments and confound previously held certainties. This explains how we laugh in bad taste, and as a way of coping with the disappointment of our lived realities. Can we also ask whether anyone really thinks Windows or Mac OS X are serious operating systems in similar terms? Or indeed, that their free software alternatives are rendered as jokes too (to counter seriousness), and reveal their futility as they in turn become commercialized (as in the case of Ubuntu perhaps)?

The point we are making is that it is possible to draw some analogies between jokes and code, as witty utterances hidden behind the surface of normative software production and totalitarian server–client architectures that really are a joke. This issue is also explored in Inke Arns's essay, 'Read_Me, Run_Me, Execute_Me', with its subtitle 'Software and its Discontents',[10] implying that the performative dimension of code lies repressed, as with the description of the Linux kernel cited earlier,[11] and further evoking Freud's *Civilisation and its Discontents,* that capitalism was founded on the repression of the libido. In post-Marxist thinking (where the constitution of subjectivity is considered more fully), something similar can be detected in the way that repression is considered to be socially determined and can only be released by freeing productive desire, perhaps expressed as nervous laughter. Drawing these threads together in the 1970s, Herbert Marcuse's *Eros and Civilisation* and, to a greater extent, Gilles Deleuze and Félix Guattari's *Anti-Oedipus: Capitalism and Schizophrenia*, argue

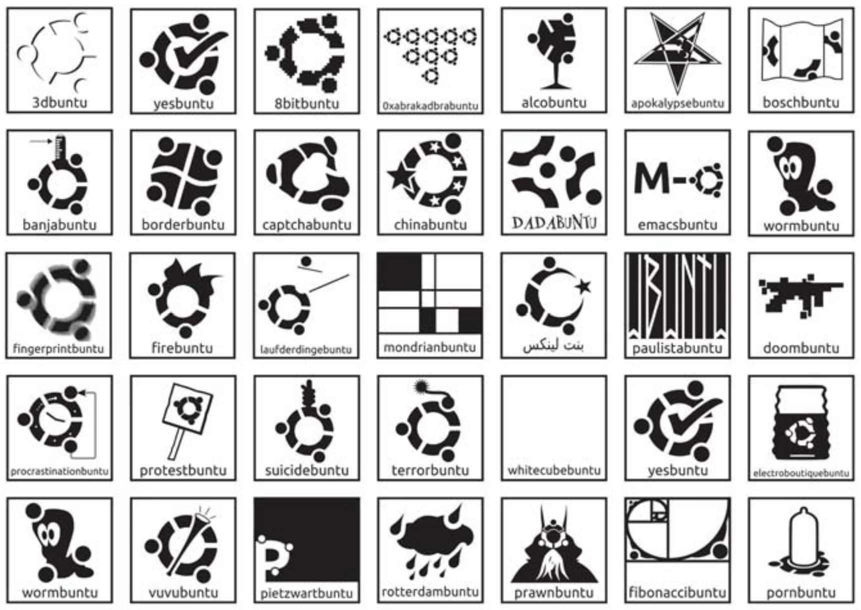


Figure 7.1 Sample of logos from Gordan Savičić and Danja Vasiliev's *120days of *buntu* (2011)

Source http://120buntu.com/

for the liberatory potential of desiring-production.[12] So in this way the release of free software can be understood as less an imaginary force based on lack (as in Freud), and more a real, productive force, as a neat example of a 'desiring-machine.[13] A good example of this is perhaps Gordan Savičić and Danja Vasiliev's project *120days of *buntu* which proposes 120 modified humorous and useless Ubuntu Operating Systems.[14] By making playful reference to de Sade's *The 120 Days of Sodom,* the suggestion is that alternative operating systems might be able to liberate desire, in excess of the masochistic desires of free/libre software development in which Ubuntu is seen to operate too close to normativity.[15]

Similarly, Franco 'Bifo' Berardi detects the fundamental political struggle between machines for liberating desire and mechanisms of control over the imaginary.[16] Thus, to Berardi the various liberatory strategies such as refusal of work, the invention of temporary autonomous zones, free software initiatives and so on offer 'dynamic recombination'.[17] Therefore, and further developing the analogy between repression and the performativity of software, free software development (where the code is made openly available/shared) might offer therapeutic assistance in putting the programmer in touch with his/her, and indeed culture's, sublimated desires repressed under proprietary software development.[18] If normative software could be thought of as software without a body, might desiring-production be explored through free software development with a body? As Christopher M. Kelty describes it, 'geeks' share a social imaginary about the production of actually existing alternatives, and as such the free software movement is an example of a 'recursive public', capable of creating and modifying the domain or platform through which they act.[19] In other words, they share a culture through which their repressed desires find release in the public domain and body politic.

The ideology of the Free Software Foundation clarifies this point with a humorous play on the ambiguity of freedom in 'you should think of "free" as in "free speech", not as in "free beer"'.[20] Commercial (free market) interests seek to assert their control over free software by replacing the word 'free' with the phrase 'open source', placing emphasis on visibility of code rather than freedom to share.[21] Moreover, the broader ideological issues are evident in the parallel narratives of the development of free and open-source software development. On the one hand, there is free software referring back to the 1980s when software freedom was meant in resistance to proprietary software, and on the other, open source that emanated from arguments about its economic benefits and in parallel to free market thinking. Furthermore, like the joke from Žižek, the futility of progressive alternatives like Ubuntu run the risk of missing the mark unless the broader political issue of the ability to modify the domain or platform through which these desiring-practices are enacted is also considered.

That freedom of speech relates to free software (at least according to the Free Software Foundation), and not free beer, may be one of the analogies that leads commentators (such as Mackenzie) to discuss the performative dimension of software and its relation to speech act theory, making reference to John Langshaw Austin's *How To Do Things With Words*.[22] Although the analogy between program code and speech acts has become rather commonplace since it was made by Terry Winogrand and Fernando Flores in 1987,[23] and is perhaps left a little general here, the point is to emphasize the ways in which programmers express themselves through the witty manipulation of layers of representation, including symbols, then words, language and notation. But clearly code is a special kind of language, and one that can automatically enable and disable certain kinds of actions.[24] It does this through its special ability to convert action to language and this is where the joke lies.

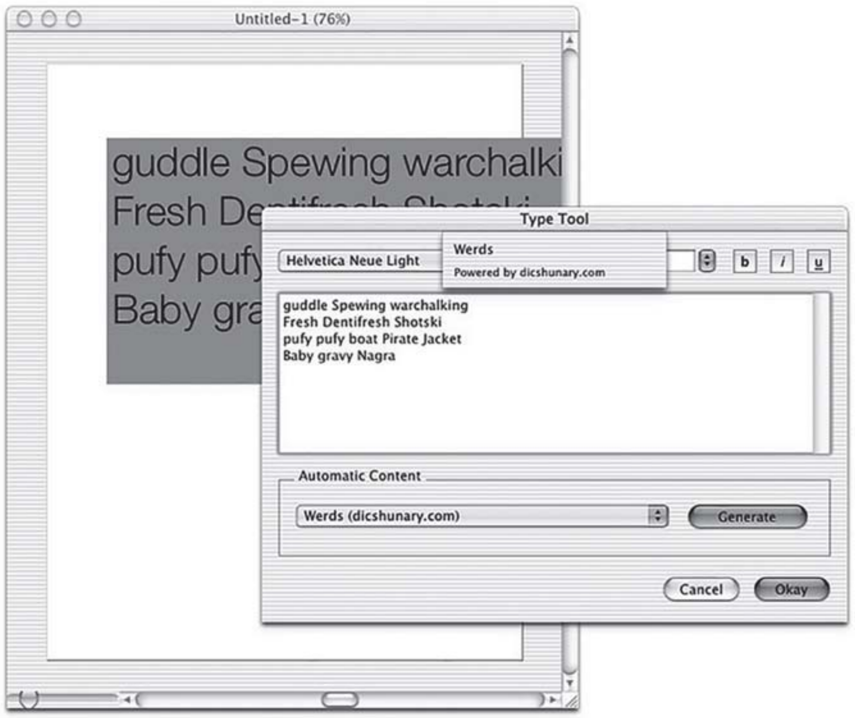
Also drawing on Austin, Virno refers to linguistic innovation as: 'how to do new things with words', where words constitute an action in and of themselves – like program code in as much as when it is interpreted it says something and does something at the same time. Virno's interest is in the ability of the human animal to execute 'innovative actions' capable of modifying 'consolidated norms'.[25] A key reference underpinning this is Noam Chomsky's description of the apparatus of power as repressing the innate creativity of verbal language, and to Aristotle's description of contingency at the heart of our use of language (in *Ethics*),[26] as well as the more general point that intellectual and linguistic labour are no longer separated from general conditions of informational capitalism. In Virno's opinion, rhetorical persuasion and the concept of the public sphere in which speech is paramount, demonstrate the ability of language to establish social relations between a 'mass of speakers', that is necessarily shared and collective.[27] He emphasizes that the 'language system is a social fact' wherein the human animal is 'ready made for language, but not actually in possession of it' until it starts interacting in the social realm.[28] This is part of early years development (the 'mirror stage', in Lacanian terms) but remains evident in every utterance made thereafter. For Virno, this confirms the biopolitical dimension of the human animal in the world, and the social importance of language and the sharing of codes.

According to Virno, underpinning political possibilities is the simple fact that the human animal is capable of modifying its forms of life.[29] This is what makes for innovation in the general sense that newly invented forms might diverge from established rules and perceived norms – based on the Chomskian innate creativity referred to above, and perhaps also to the idea that jokes can be understood as hard-wired (in as much as creativity relates to playing with language). And this is where jokes also figure as an example of how humans diverge from social norms, how 'linguistic animals give evidence of an unexpected derivation from their normal praxis', as Virno puts it.[30] For him, witty utterances are similar to the performative utterances that Austin described, where words constitute an action in and of themselves.[31] But as already emphasized, the point for Virno is not the content of jokes, which might poke fun at social norms, hierarchies or the ruling order, for such jokes tend to obscure what is important: the apparatus or the '*logicolinguistic resources* that jokes utilize'.[32] His argument is that innovative action uses these resources like a toolbox or library. In this way, it produces ambivalences: oscillating between the 'determined *rule* and the *regularity* of species-specific forms of conduct'.[33] Such contradictory factors characterize the social character of the human species and its innovative force, despite its repression by power structures. In other words, jokes demonstrate innovative techniques and the possibilities for transforming the linguistic operating system. This happens in two main ways according to Virno: first by demonstrating how divergences in following rules often result in changing the rule itself (put differently the application of the norm also contains surprises, and situations where the rule is broken and justified in terms of exceptional circumstances[34]); and secondly, through the incorrect use of semantic ambiguity, an 'error' (or glitch).[35] The rules are not only there to be broken, but applied differently, adapted and modified, and ultimately transformed. So how does software demonstrate similar possibilities at the level of code, and where the modifications undermine normalized behaviour?

Code jokes

It would be relatively easy to mention the use of parody in this connection to software, but perhaps this is too predictable under present conditions and its effectiveness thereby questionable in a similar manner to a political joke that fails to register its wider effects once executed. So although an example like Gordan Savičić's *sing_slavoj_sing* might poke fun at the seriousness of Žižek's philosophizing, by hacking *billy the fish* and replacing the soundchip with a good quote or two, this is not the kind of joke that we wish to emphasize for our argument as it does not present an intervention in terms of the apparatus of language.[36] If humour is somewhat hard-wired, then jokes are an inevitable component of the structures of language where humans demonstrate their innate ability to innovate new forms. But can code jokes be reduced to their functional aspects in this way, as procedures that are open to recombination as language is more generally? Our concern is about jokes at the level of changing rules and the use of semantic ambiguity, in keeping with Virno's emphasis thus far.

Such tendencies can be detected in the example of Signwave's *Auto-Illustrator*, an experimental, semi-autonomous, generative software artwork that includes a wide range of generative and procedural techniques, packaged as a fully functioning parody of Adobe's vector graphics application *Illustrator*.[37] Humour is an important part of the software package as indicated, not least, by the release date of version 1.2 on April Fool's Day, in 2003. The new release included parody plug-ins that serve to question how contemporary software should 'behave',



guddle Spewing warchalki
Fresh De
pufy pufy
Baby gra

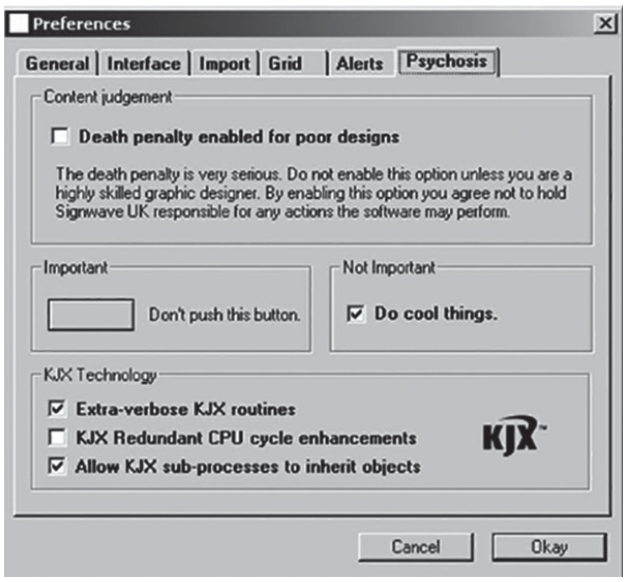
Helvetica Neue Light

Werds
Powered by dicshunary.com

guddle Spewing warchalking
Fresh Dentifresh Shotski
pufy pufy boat Pirate Jacket
Baby gravy Nagra

Automatic Content

Werds (dicshunary.com)



Content judgement

☐ Death penalty enabled for poor designs

The death penalty is very serious. Do not enable this option unless you are a highly skilled graphic designer. By enabling this option you agree not to hold Signwave UK responsible for any actions the software may perform.

Important

Don't push this button.

Not Important

☑ Do cool things.

KJX Technology

☑ Extra-verbose KJX routines
☐ KJX Redundant CPU cycle enhancements
☑ Allow KJX sub-processes to inherit objects

KJX™

Figures 7.1 and 7.2 Screenshots of Signwave's *Auto-Illustrator* (2001): 'Dicshunary' spelling tool and 'Psychosis' preferences option

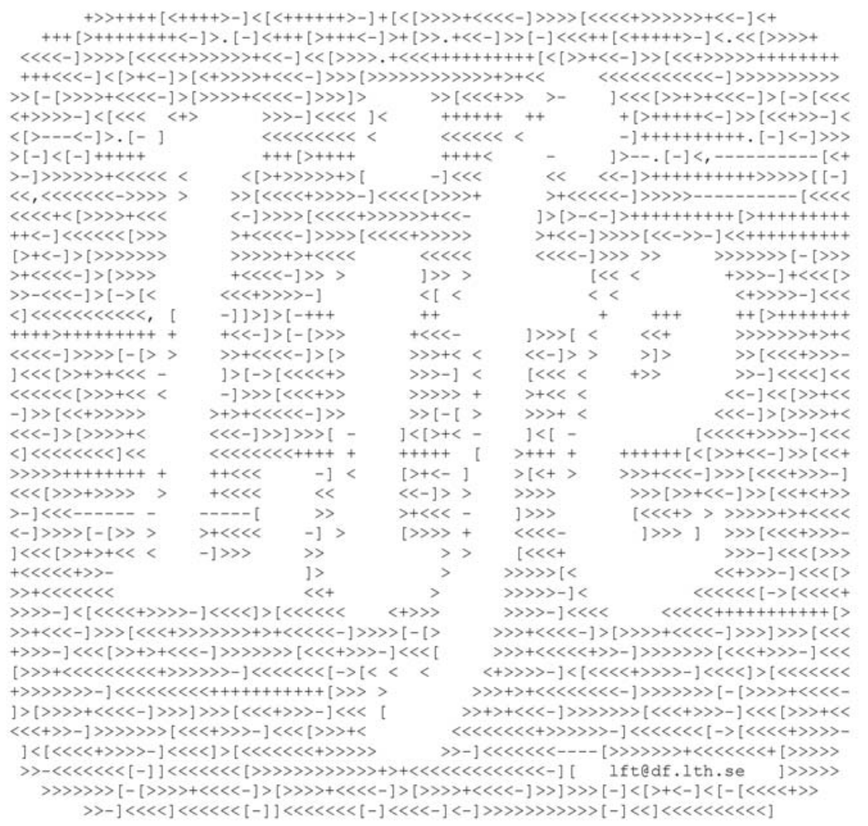
Source http://swai.signwave.co.uk

including one function that would shut the computer down with no notice if the software 'decided' the design was unworthy. The user's expectations were challenged by presenting tools and functions that do not conform to expectations of what software usually does as a tool. Perhaps most controversial was the terms of use which insisted that designs using the software were co-credited to the author and the software company. This caused some outrage by users who assumed the company were overstepping their proprietary rights and imposing measures beyond a joke. Yet, to Signwave, this seemed to be an entirely logical statement which recognized the difficulties of making any claim of single authorship, and moreover exemplified the principle that the making of software can be regarded as an artwork in itself. Thus Signwave extended some of the ambiguities built into software production into a playful form that relates to the way social relations are organized with normative software development. If *Auto-illustrator*'s satire of Adobe's *Illustrator* operates at the level of code in a general sense – in the domain of intellectual property within the legal system, itself a system of rules, codes and speech acts – could this also work in the domain of source code?

To repeat the Virno formulation, jokes are identified as operating in two significant ways: first by demonstrating how divergences in following syntactical rules often generate a change in the rule itself; and secondly, through the ironic use of semantic ambiguity. This perhaps happens in a general way with *Auto-illustrator*, but how do these ideas translate to code more precisely? In this sense, it could be said that we wish to crack the source code of jokes by examining the concept of interpretation in more detail. The interpreter is a codification of the language in which the source code is expressed, or, in other words, a partially evaluated computational process.[38] The source code is not a program on its own, rather it is a replaceable component of a larger computation. The relationship between source code and interpreter is recursive; the rules behind an interpreter are themselves implemented in source code, requiring another interpreter. If we follow these recursive layers of interpretation we find hardware microcode that mediates between the discrete digital world and our continuous physical world.

With few exceptions, mainstream programming languages are Turing-complete. This means it is possible to write a program in any of the languages that interprets any other computer language. Therefore the rules of any language can be changed simply by writing an interpreter for another language within it; and thereby the scope for breaking rules is boundless. Turing-completeness extends beyond mainstream language into some surprising places. Simple cellular automata, including the well-known *Game of Life* by John Conway and Stephen Wolfram's *Rule 110*,[39] even some configuration files such as that of the 'sendmail' electronic mail transport software, have turned out to be Turing-complete. The fact that Turing-completeness comes so easily to these invented languages, allowing any symbolic constraint to be broken at will, provides a programmer's playground with glimpses of the infinite.

With this emphasis on breaking rules, it is no surprise that source code humour centres on interpretation of code. *Esoteric* programming languages are those which take a humourous approach to language design, challenging the norms of source code interpretation. For example, obfuscated programs written in the brainfuck language consist entirely of punctuation, each of the eight characters '><+-.,[]' representing a single elementary operation. Brainfuck is Turing-complete, and so can be used to write any program in theory, but in practice it is extremely difficult to write any program in it at all. Below is *The Game Of Life* implemented in brainfuck.[40]



```
       +>>++++[<++++>-]<[<++++++>-]+[<[>>>>+<<<<-]>>>>[<<<<+>>>>>>+<<-]<+
   +++[>++++++++<-]>.[-]<+++[>+++<-]>+[>>.+<<-]>>[-]<<<++[<+++++>-]<.<<[>>>>+
 <<<<-]>>>>[<<<<+>>>>>>+<<-]<<[>>>>.+<<<++++++++++[<[>>+<<-]>>[<<+>>>>>++++++++
 +++<<<-]<[>+<-]>[<+>>>>+<<<-]>>>[>>>>>>>>>>>>>+>+<<     <<<<<<<<<<<-]>>>>>>>>>>
>>[-[>>>>+<<<<-]>[>>>>+<<<<-]>>>]>      >>[<<<+>>  >-    ]<<<[>>+>+<<<-]>[->[<<<
<+>>>>-]<[<<<  <+>      >>>-]<<<< ]<     ++++++  ++       +[>+++++<-]>>[<<+>>-]<
<[>---<-]>.[- ]         <<<<<<<<<< <      <<<<<< <         -]++++++++++.[-]<-]>>>
>[-]<[-]+++++           +++[>++++        ++++<     -     ]>--.[-]<,----------[<+
>-]>>>>>>+<<<<< <     <[>+>>>>>+>[      -]<<<      <<   <<-]>++++++++++>>>>>[[-]
<<,<<<<<<<->>>> >    >>[<<<<+>>>>-]<<<<[>>>>+      >+<<<<<-]>>>>>----------[<<<<
<<<<+<[>>>>+<<<      <-]>>>>[<<<<+>>>>>>+<<-      ]>[>-<-]>++++++++++[>+++++++++
++<-]<<<<<<[>>>      >+<<<<-]>>>>[<<<<+>>>>>      >+<<-]>>>>[<<->>-]<<++++++++++
[>+<-]>[>>>>>>>      >>>>>+>+<<<<      <<<<<      <<<<-]>>> >>     >>>>>>>[-[>>>
>+<<<<-]>[>>>>       +<<<<-]>> >       ]>> >            [<< <        +>>>-]+<<<[>
>>-<<<-]>[->[<      <<<+>>>>-]          <[ <             < <           <+>>>>-]<<<
<]<<<<<<<<<<<, [    -]]>]>[-+++        ++               +    +++     ++[>+++++++
++++>++++++++++ +    +<<-]>[-[>>>      +<<<-      ]>>>[ <    <<+      >>>>>>>>+>+<
<<<<-]>>>>[-[> >    >>+<<<<-]>[>      >>>+< <    <<-]> >    >]>      >>[<<<+>>>-
]<<<[>>+>+<<< -     ]>[->[<<<<+>      >>>-] <    [<<< <    +>>       >>-]<<<<]<<
<<<<<<[>>>+<< <     -]>>>[<<<+>>      >>>>> +    >+<< <             <<-]<<[>>+<<
-]>>[<<+>>>>>      >+>+<<<<<-]>>      >>[-[ >    >>>+ <            <<<-]>[>>>>+<
<<<-]>[>>>>+<      <<<-]>>]>>>[ -    ]<[>+< -    ]<[ -           [<<<<+>>>>-]<<<
<]<<<<<<<<]<<      <<<<<<<<++++ +    +++++ [   >+++ +    ++++++[<[>>+<<-]>>[<<+
>>>>>+++++++++ +    ++<<<     -] <    [>+<- ]    >[<+ >    >>>+<<<-]>>>[<<<+>>>-]
<<<[>>>+>>>>  >    +<<<<     <<      <<-]> >    >>>>       >>>[>>+<<-]>>[<<+<+>>
>-]<<<------ -    -----[     >>      >+<<< -    ]>>>       [<<<+> > >>>>>+>+<<<<
<-]>>>>[-[>> >    >+<<<<    -] >     [>>>> +    <<<<-       ]>>> ]  >>>[<<<+>>>-
]<<<[>>+>+<< <    -]>>>     >>           > >    [<<<+              >>>-]<<<[>>>
+<<<<<+>>-                   ]>           >     >>>>>[<             <<+>>>-]<<<[>
>>+<<<<<<<                   <<+         >      >>>>>-]<          <<<<<<[->[<<<<+
>>>>-]<[<<<<+>>>>-]<<<<]>[<<<<<<    <+>>>      >>>>-]<<<<     <<<<<++++++++++++[>
>>+<<<-]>>>[<<<+>>>>>>>+>+<<<<<-]>>>>[-[>     >>>+<<<<-]>[>>>>+<<<<-]>>>]>>>[<<<
+>>>-]<<<[>>+>+<<<-]>>>>>>>[<<<+>>>-]<<<[     >>>+<<<<<+>>-]>>>>>>>[<<<+>>>-]<<<
[>>>+<<<<<<<<<+>>>>>>-]<<<<<<<[->[< <  <     <+>>>>-]<[<<<<+>>>>-]<<<<]>[<<<<<<<
+>>>>>>>-]<<<<<<<<<++++++++++[>>> >        >>>+>+<<<<<<<<<-]>>>>>>>[-[>>>>+<<<<-
]>[>>>>+<<<<-]>>>]>>>[<<<+>>>-]<<< [       >>+>+<<<-]>>>>>>>[<<<+>>>-]<<<[>>>+<<
<<<+>>-]>>>>>>>[<<<+>>>-]<<<[>>>+<        <<<<<<<<+>>>>>>-]<<<<<<<[->[<<<<+>>>>-
 ]<[<<<<+>>>>-]<<<<]>[<<<<<<<+>>>>>      >>-]<<<<<<<----[>>>>>>>+<<<<<<<+[>>>>>
 >>-<<<<<<<[-]]<<<<<<<[>>>>>>>>>>>>+>+<<<<<<<<<<<<<-][   lft@df.lth.se   ]>>>>>
   >>>>>>>[-[>>>>+<<<<-]>[>>>>+<<<<-]>[>>>>+<<<<-]>>]>>>[-]<[>+<-]<[-[<<<<+>>
       >>-]<<<<]<<<<<<[-]]<<<<<<<[-]<<<<-]<-]>>>>>>>>>>>[-]<<]<<<<<<<<<<]
```

A comment on the left side of the web page where the program is posted, simply states 'get a Life;)'.

The tortuous nature of brainfuck escapes the world of computation to be wrought on the human body as bodyfuck, a brainfuck interpreter using computer vision techniques to map from bodily gestures to the brainfuck instructionset (like a desiring-machine). The brainfuck demo shows how much physical exertion is required to produce a short sequence of symbols,[41] and the programmer uses his/her body in a more overt manner than previously required. To take another example, befunge is an esoteric language that breaks the usual downward direction of interpretation through a program to create two-dimensional syntax. This is done using punctuation, as in brainfuck but in a more understandable way; each of the four instructions '^>v<' represent graphical arrows, which change the direction of control flow as you might expect. The question mark character '?' changes the direction in a random direction. The instruction set goes beyond this, and is again Turing-complete, but by following the arrows we can already understand the operation of the following program, which, when read starting from the top left-hand corner, outputs a random number from 0 to 9:

```
vv  <      <
    2
    ^  v<
 v1<?>3v4
    ^   ^
>  >?>  ?>5^
    v   v
 v9<?>7v6
    v  v<
    8
 .  >  >   ^
^<
```

Despite being Turing-complete these interpreters appear useless, and some esoteric languages seem to exist simply as 'in jokes' for geeks, having unimplementable instruction sets. In the case of IRP (Internet Relay Programming) there is no formal instruction set at all, the interpreters are human participants in an internet chat room, a running joke since 2005. Below is the obligatory 'Hello, World!' example:

<GregorR>	Please say 'Hello, World!'
<jix>	Hello, World!

The likelihood of an IRP program being interpreted as you wish is improved if you are polite, however this does not always work:

<GregorR>	Please, write the lyrics to the song 99 Bottles of Beer on the Wall.
<memonic>	go to hell

Recursion is possible, up to a point:

<CakeProphet>	Could someone please ask someone to repeat this request?
<pikhq>	Could someone please repeat the previous request?
<RodgerTheGreat>	Could someone please ask someone to repeat this request?

In recasting English as a machine language and humans as interpreters, we are given the opportunity to examine the relationship between performative phrases and computer code in some detail. In this way, we might begin to understand that jokes can be interpreted in multiple ways which reflect the complexity of human and machine logic that moves beyond simple amusement. Indeed all jokes express a purpose. As in Virno's earlier descriptions, this is where innovative techniques are demonstrated that diverge from established conventions by changing and breaking rules, and playing with ambiguities related to code. This is where other possibilities reside.

But is software development taken too seriously when it is a joke all along? Indeed software development is in danger of losing its sense of humour altogether as it becomes more and more standardized and packaged. For instance, with service-based platforms access to source code is no longer possible, and the differentiation between files, software and network services evaporates altogether. This is an apt description of the Apple iOS paradigm of software development, where users of the Apple iPad and iPhone are allowed only restricted access to programming language interpreters, and software licenses favoured by the Free Software Foundation are forbidden. Access to humour is denied and the example confirms that proprietary logic is a serious business of repression. In Berardi's terms, interpretation has become schizophrenic (like fast speech), and the relations between metaphors and things, representation and life, have become thoroughly confused. This is particularly evident when it comes to language that is more and more influenced by machines, leading to a situation where the learning of language and affectivity have been separated.[42] On the contrary, what needs to be rediscovered are forms of happiness and laughter tied to collective formations: the sharing of code that liberates desire and mechanisms of control over the imaginary from the serious business and sense of determinism that is normally associated with code.[43]

The reference to Virno's work on jokes draws attention to their function in relation to 'innovative action' in the public sphere. In the case of the title of this chapter, the use of 'not' as prefix to Torvald's title (in 'not just for fun') is in keeping with Virno's comments that the system of language both 'does' negation (by identifying what something is not), and 'is' negation (in as much as it can only signify something): 'The negation, or something that language *does*, is understood, above all, as something that language *is*'.[44] He is speculating here on a non-representational form of politics, and despite recognizing the sovereign forces that restrain such abilities, concludes that humans are capable of adapting themselves and their circumstances in parallel to their linguistic abilities and possibilities for innovative action. We have attempted to consider the execution of program code in similar terms; that the potential for divergent forms comes into existence through social interactions and modifications. Rules are not just there to be broken, but transformed altogether through negation. For instance, the function of humour (to Žižek) lies in its ability to invert common sense. Code jokes remind us of the possibilities of non-deterministic interpretation in this inverse way, as deviations from the conventions of coding as formal expression. What we mean to stress is that having fun with software does not simply encourage people to work with computers (and distract them from the hard work that is unavoidable when programming) but offers a way to rethink political possibilities in public, and to reimagine the seriousness of normative visions of life that code otherwise implies.

Notes

1 Virno, Paolo, *Multitude: Between Innovation and Negation*, trans. Isabella Bertoletti, James Cascaito and Andre Casson (Los Angeles: Semiotext(e) Foreign Agents, 2008).

2 This issue of fun is also something that Christopher M. Kelty raised in his talk 'No Fun. Work, Labor, Action in Free Software', presented at 'The Internet as Playground and Factory Conference', Eugene Lang College, The New School, New York (12–14 November 2009), http://digitallabor.org/. His presentation slides

can be seen online at http://www.scribd.com/doc/22394057/No-Fun-Slides (all accessed 12.12.2013).

3 Cited in Bogost, Ian, 'An Alternative to Fun', in *Unit Operations* (Cambridge, MA: MIT Press, 2006), 118.

4 Torvalds, Linus, *Just for Fun: The Story of an Accidental Revolutionary* (New York: HarperCollins, 2001), http://en.wikipedia.org/wiki/Just_for_Fun (accessed 12.12.2013).

5 Stallman, Richard, *Free Software, Free Society: Selected Essays of Richard M. Stallman*, ed. Joshua Gay (Boston, MA: Free Software Foundation, 2002), 15.

6 Mackenzie, Adrian, 'The Performativity of Code: Software and Cultures of Circulation', in *Theory, Culture & Society*, 22(1): 13.

7 Freud, Sigmund, *The Joke and Its Relation to the Unconscious* (London: Penguin, 2002).

8 Dillon, Brian and Simon Critchley, 'Very Funny: An Interview with Simon Critchley', in *Laughter*, *Cabinet*, 17 (spring 2005), http://www.cabinetmagazine.org/issues/17/dillon.php (accessed 12.12.2013).

9 In this connection, it is well worth watching his performance in a clip from YouTube where he tells an old Russian joke, http://www.youtube.com/watch?v=XEnkDEgALGI (accessed 12.12.2013).

10 Arns, Inke, 'Read_Me, Run_Me, Execute_Me: Software and Its Discontents, or: It's The Performativity of Code, Stupid', in *Read_Me: Software Art & Cultures – Edition 2004*, (eds) Olga Goriunova and Alexei Shulgin, 176–93 (Aarhus, Denmark: Digital Aesthetics Research Centre, Aarhus University, 2004).

11 See note 6.

12 Deleuze, Gilles and Guattari, Félix, *Anti-Oedipus: Capitalism and Schizophrenia*, trans. Robert Hurley, Mark Seem and Helen R. Lane (London: Athlone, 1990).

13 Ibid.

14 Savičić, Gordan and Vasiliev, Danja,*120 days of *buntu* (2011), http://120buntu.com/ (accessed 12.12.2013).

15 See Cox, Geoff, 'Notes on 120 days of *buntu', in *World of the News*, (eds) Geoff Cox and Christian Ulrik Andersen (Berlin/Aarhus: transmediale/DARC, 2012), 9.

16 Berardi, Franco 'Bifo', *Precarious Rhapsody: Semiocapitalism and the Pathologies of the Post-alpha Generation* (London: Minor Compositions, 2009). His performative reading of the source code of the 'I Love You' virus resonates with this description; he read the source code of the virus at the D-I-N-A (Digital Is Not Analog) digital art festival in 2001, http://www.digitalcraft.org/iloveyou/loveletter_reading.htm (accessed 12.12.2013).

17 Ibid., 72.

18 In French, free software is known as 'libre' software. Although it does not have an analogue in English, this unambiguous term evokes a libertarian attitude.

19 Kelty, Christopher M. *Two Bits: The Cultural Significance of Free Software* (Durham, NC: Duke University Press, 2008).

20 See http://www.gnu.org/philosophy/free-sw.html. This was made into more of a joke by the Danish artists Superflex in their project Free Beer, in which they share recipes for free beer, http://www.superflex.net/projects/freebeer/ (both accessed 12.12.2013).

21 Releasing and sharing source code therefore represents a number of ambiguities relating to trust, cost, liberty, making free but making money on the stock market instead, a belief in open standards or a cynical business move to capitalize on free labour.

22 Austin, John Langshaw, *How to Do Things with Words* (Cambridge, MA: Harvard University Press, 1962). For an elaboration of this, see Cox, Geoff and Alex McLean, *Speaking Code: Coding as Aesthetic and Political Expression* (Cambridge, MA: MIT Press, 2011).

23 Winograd, Terry and Flores, Fernando, *Understanding Computers and Cognition: A New Foundation for Design* (Reading, MA: Addison-Wesley, 1987).

24 Chun, Wendy, *Programmed Visions: Software and Memory* (Cambridge, MA: MIT Press, 2011), 27.

25 Virno, *Multitude: Between Innovation and Negation,* 20.

26 Ibid., 13.

27 Ibid., 46.

28 Ibid., 47.

29 Ibid., 69.

30 Ibid., 72.

31 Ibid., 85.

32 Ibid., 165.

33 Ibid., 166.

34 The so-called 'state of exception', invoking Carl Schmitt, to describe the way the state legitimates the breaking of its own legal rules in exceptional circumstances. According to Giorgio Agamben, this has become the working paradigm of modern government.

35 Ibid., 73, 74.

36 *sing_slavoj_sing* was developed for DEVICE-ART Zagreb, by fleshgordo (Gordan Savičić) in 2006. See http://www.youtube.com/watch?v=D9FXyr-LLeI (accessed 12.12.2013).

37 See http://swai.signwave.co.uk/ (accessed 12.12.2013). See Signwave, *Auto-Illustrator Users Guide* (first written 2001), published to coincide with the

boxed set as part of the *Generator* exhibition, Spacex 2002–03, and touring in the UK.

38 Futamura, Yoshihiko, 'Partial Evaluation of Computation Process – An Approach to a Compiler-compiler', in *Higher-Order and Symbolic Computation*, 12(4) (1999): 381–91, http://portal.acm.org/citation.cfm?id=609205 (accessed 12.12.2013).

39 John Conway's *Game of Life* is a cellular automaton devised in 1970, consisting of a collection of cells which, based on a few mathematical rules, can live, die or multiply. See http://www.bitstorm.org/gameoflife/ (accessed 12.12.2013). Similarly Stephen Wolfram's *Rule 110* is an elementary cellular automaton with interesting behaviour on the boundary between stability and chaos that is considered to be Turing-complete, meaning that in principle, any calculation or computer program can be simulated using this automaton. See http://www.wolframalpha.com/input/?i=rule+110 (accessed 12.12.2013).

40 Akesson, Linus, *The Game of Life* implemented in brainfuck, http://www.linusakesson.net/programming/brainfuck/index.php (accessed 12.12.2013).

41 See http://www.nikhanselmann.com/?/projects/bodyfuck/ (accessed 12.12.2013).

42 Berardi, *Precarious Rhapsody*, 9.

43 As our smiles are clearly not simply fixed like George Maciunas' *Flux Smile Machine* (1971), a device that forces the shape of the user's mouth into a smile. This work was the reference point for the exhibition *Smile Machines*, curated by Anne-Marie Duguet, for the Transmediale Festival for Digital Culture, in 2006.

44 Virno, *Multitude: Between Innovation and Negation,* 50.

8

Fun is a Battlefield: Software between Enjoyment and Obsession

Wendy Hui Kyong Chun and Andrew Lison

As this volume makes clear, fun has been fundamental to the emergence, solidification and spread of software: from higher-level programming languages, designed to make programmers masters of, rather than servants to, their machines, to graphical user interfaces (GUIs), which transform computers from command-based instruments of torture into user-friendly universal mediums, to Web 2.0 applications such as Foursquare, which make self-surveillance a game. Indeed, fun would now seem synonymous with software, or at least gameware: in 2010, consulting gurus stressed 'funware' – game mechanics such as leaderboards, points, levels, etc. – as *the* way to generate user loyalty submission.[1] Fun makes software more than just software; it makes it a problem-driven adventure or a game, at all levels of engagement. This in turn makes fun more than fun: software use and programming (increasingly similar activities, given ever more abstracted programming languages and more efficient AI) become productive, often exploited yet freely given forms of labour;[2] they also become subjects of obsession.

Fun, however, has always been more than fun; it has always been an economic relation. Etymologically, to be 'funned' was to be cheated, the victim of a joke.[3] One 'funned' another out of money. Fun, in this sense, is always 'more' – an illicitly generated surplus. The question is thus not why are fun and exploitation now related? But rather, why do we think they're not? Current understandings of fun as a harmless diversion erase the victor–victim relation emphasized in early uses of 'fun' as a verb. Fun as 'just fun' assumes that everyone is the subject of, rather than subject to, fun. It elides the position of a loser, or perhaps more precisely, it places losers is in the position of now-toothless victors. This erasure benefits those who provide the grounds for 'fun', since those who have fun do so while being funned.

In many ways, this seems to perfectly encapsulate what Tiziana Terranova has called 'free labour': activities such as the production of unremunerated user content and free software development that are 'simultaneously voluntarily given and unwaged, enjoyed and exploited'.[4] Linking free labour to the move from factory-based to immaterial economies, which blur distinctions between producers and consumers, she sees free labour as 'the moment where… knowledgeable consumption of culture is translated into productive activities'.[5] Crucially, capitalism does not simply suck the blood of free labour, for global capitalism is the field from which free labour flows. This does not mean, however, that capitalism entirely controls free labour: although capitalism sets the conditions for free labour and filters the valuable from the invaluable, free labour for Terranova, as a form of mass knowledge, always threatens to move elsewhere, to be the uncontrollable basis for a common passion. Drawing from and extending her analysis, we view fun as neither 'oppressive' (bad in a bad way) nor 'subversive' (bad in a good way, as Anna Fisher has so wonderfully argued).[6] Fun, we argue, is a site of struggle – through fun, the free and unfree are linked together.

Stressing the subject–object relation highlighted in archaic uses of fun, we maintain that fun is a battlefield: a ticklish space of struggle and conflict, of sometimes real or imagined victory. However, our point is not simply to condemn fun and return it to some originary meaning, but rather to understand the potential and dangers of fun, and to think through how these might be the same thing. Intriguingly, fun, as enjoyment, is bounded by the concept of obsession: fun, that is, is no longer fun when it's so fun it's not, when the 'more' becomes too much. Obsession highlights fun as a form of loss, but does so by moving actions directed at someone to more-or-less self-inflicted injuries. The term 'obsession', defined as 'the action of besieging a place', curiously disappeared in the sixteenth century, only to be resuscitated in the nineteenth, when it became a 'compulsive interest or preoccupation'.[7] A person obsessed, like the melancholic, is one possessed by the other within oneself, by some thought, impulse or image that will not let go. Programming and gaming seem to play with the boundary between fun and obsession. A hacker becomes obsessed with the problem to be solved, but he/she also risks becoming the victim of obsession: the classic example, the video-game-obsessed programmer who never finishes the task at hand and thus, becoming addicted, switches from 'creator' to 'user'.[8]

The fuzzy boundary between fun and addiction highlights the fact that the fun of programming is not simply linked to the possession of power, but also, as we explain further, the experience of the limitations of one's power within a system that promises unlimited power. To be clear, we are neither celebrating obsession and addiction nor arguing that we need to always stay on track. Rather, we are arguing that fun and software thrive on the constant crossing of the boundary between enjoyment and obsession. If fun has increasingly motivated the rise of software, our question is: are 'freaks' running the world through fun, or, in an immaterial, affective economy, is fun running the world through freaks, those motivated, through their own experience of enjoyment, to obsession? Or both? If there is a kind of surplus in fun, we want to think of it as lying not in the simple act of gratification, but rather in something like the dialectic between engagement and oppression, enjoyment and obsession, hoaxed and hoaxer. Or, drawing on Sara Ahmed's work on emotions, in which she argues that emotions are not something one 'has', but rather

> they create the very effect of the surfaces and boundaries that allow us to distinguish between an inside and an outside in the first place…it is through emotions, or how we respond to objects and others, that surfaces and boundaries are made[9]

we want to understand how the experience of fun, including its 'dangers', is central to the emergence of programmers and users. Any consideration of fun as a potentially liberatory force will therefore have to consider the economics of affect both from the position of Ahmed's cultural politics as well as Terranova's 'free labour'.[10]

To think through these questions, then, we present a series of cases, from a potted history of early programming to obfuscated C, from 1990s rave culture to software company office slides. In the spirit of this collection, the form of this chapter is itself experimental: a dialogue rather than a monologue.

Programming is suffering and therefore fun

How did programming become fun? More to the point: how did programming become programming?

Software, as one of us has argued in more detail elsewhere, was not foreseen: the engineers building what would become the first computers did not plan for software.[11] The first 'programmer' – the 'master programmer' – was machinic: a component of the ENIAC (widely considered the first working digital electronic computer) that determined the sequence of the calculation. Computers, back then, were human, and those later acknowledged as the first programmers were former computers: the so-called ENIAC girls (later the more politically correct 'women of the ENIAC'), Kathleen McNulty, Mauchly Antonelli, Jean Jennings Bartik, Frances Snyder Holberton, Marlyn Wescoff Meltzer, Frances Bilas Spence and Ruth Lichterman Teitelbaum. These women engaged in 'direct programming', wiring the machine for each problem, mastering the master programmer so processes could loop and repeat. The reclamation of these women as the first programmers in the mid-to-late 1990s was love at last sight: by then, not only had women 'coders' been pushed almost completely out of the workplace, the 'direct' knowledge of the machine they possessed had been compromised by the increasing automation of programming. The move to humanize programming and to mechanize computing coincides with 'automatic programming', a phenomenon that sought to make programming more 'fun' by increasingly removing the programmer from wearisome machine processes: higher-level languages helped make programming problem- and object-oriented, rather than focused on the mind-numbing minutiae of machine languages. This was also a blatant move to disempower the high priests of machine-language programming, who, according to John Backus, the developer of FORTRAN, viewed themselves as possessing super-human knowledge.[12] Tellingly, this language of high priests and wizards has not disappeared with higher-level languages, although this disempowerment has spread everywhere, allegedly for the good of the programmer and the advancement of computing. Knowledge, as John V. Guttag, a 'pioneer' in data abstraction explains, is dangerous: '"Drink deep, or taste not the Pierian Spring", is not necessarily good advice. Knowing too much is no better, and often worse, than knowing too little. People cannot assimilate very much information. Any programming method or approach that assumes that people will understand a lot is highly risky'.[13]

So what was/is programming: drudgery or magic? Do higher-level languages offer more power or greater ignorance? What is crucial, we argue, is not to decide this debate, but rather to recognize its fundamental importance. This oscillation is not accidental, but central to the power and enjoyment – the hatred and love – of programming. An excellent early example of this is MIT professor, creator of ELIZA, and member of the famed AI Lab Joseph Weizenbaum's description of programming. At first, he contends:

> The computer programmer…is a creator of universes for which he alone is the lawgiver. So, of course, is the designer of any game. But universes of virtually unlimited complexity can be created in the form of computer programs. Moreover, and this is a crucial point, systems so formulated and elaborated *act out* their programmed scripts. They compliantly obey their laws and vividly exhibit their obedient behavior. No playwright, no stage director, no emperor, however powerful, has ever exercised such absolute authority to arrange a stage or a field of battle and to command such unswervingly dutiful actors or troops.[14]

Not coincidentally, Weizenbaum ends this quotation by referring to a battlefield and a stage: war games. The progression from playwright to stage director to emperor is telling: programming languages, like neoliberal economics, model the world as a 'game'.[15] Programming languages establish the programmer as sovereign, one for whom there is no difference between command given and command completed. The programmer says 'let there be light' and there is light. Like Faust's translation of *logos* as 'deed', software is action, so that 'in the beginning was the Word, and the Word was with God, and the Word was God'.[16]

By doing what it says, code is *logos*. Like the King's speech in Plato's *Phaedrus*, it does not pronounce knowledge or demonstrate it – it transparently pronounces itself.[17] The hidden signified – meaning – shines through and transforms itself into action. Programming languages offer the lure of visibility, readability, logical – if magical – cause and effect. As Fred Brooks argues, 'one types the correct incantation on the keyboard, and a display screen comes to life, showing things that never were nor could be'.[18] One's word creates something living. Consider this ubiquitous 'hello world' program written in C++ ('hello world' is usually the first program a person will write):

```
// this program spits out 'hello world'
#include <iostream.h>
int main ()
{
        cout << 'Hello World!';
        return 0;
}
```

The first line is a comment line, explaining to the human reader that this program spits out 'hello world'. The next line directs the compiler's pre-processor to include iostream.h, a standard file to deal with input and output to be used later. The third line, 'int main ()', begins the main function of the program;

'cout<<"Hello World!";' prints 'Hello World!' to the screen ('cout' is defined in iostream.h); 'return 0' terminates the main function and causes the program to return a 0 if it has run correctly. Although not immediately comprehensible to someone not versed in C++, this program nonetheless seems to make some sense, seems to be readable. It comprises a series of imperatives and declaratives that the computer presumably understands and obeys. When it runs, it follows one's commands and displays 'Hello World!'

It is no accident that 'Hello World!' is the first program one learns, because 'Hello World!' is easy and enjoyable: it makes us see that we can produce results immediately. This ease, according to Weizenbaum, makes programming seductive and dangerous:

> It happens that programming is a relatively easy craft to learn…And because programming is almost immediately rewarding, that is, because a computer very quickly begins to behave somewhat in the way the programmer intends it to, programming is very seductive, especially for beginners. Moreover, it appeals most to precisely those who do not yet have sufficient maturity to tolerate long delays between an effort to achieve something and the appearance of concrete evidence of success. Immature students are therefore easily misled into believing that they have truly mastered a craft of immense power and of great importance when, in fact, they have learned only its rudiments and nothing substantive at all…[19]

The quick reactions of the computer produce a feeling of power that fuels the transformation of 'immature students' into programmers. At the same time that the experience of programming models that of the sovereign, however, its seeming ease hides a greater difficulty – executability leads to unforeseen circumstances, unanticipated or 'buggy' repetitions. Indeed Weizenbaum's early description of the programmer as unfailing law-giver seems undercut by his later description of a program as a court of law, the workings of which are not visible to the programmer. The programmer, Weizenbaum stresses, is fundamentally ignorant:

> a large program is, to use an analogy of which Minsky is also fond, an intricately connected network of courts of law, that is, of subroutines, to which evidence is transmitted by other subroutines. These courts weigh (evaluate) the data given to them and then transmit their judgments to still other courts. The verdicts rendered by these courts may, indeed, often do, involve decisions about what court has 'jurisdiction' over the intermediate results then being manipulated. The programmer thus cannot even know the path of decision-making within

> his own program, let alone what intermediate or final results it will produce. Program formulation is thus rather more like the creation of a bureaucracy than like the construction of a machine of the kind Lord Kelvin may have understood.[20]

The erasure of the judicial system – of the gap between law and execution – is central to code as *logos*. As in the neoliberal model of governance, the individual programmer is thus made to feel sovereign at the same time he/she is made subject; subjected to unseen limitations, the seemingly all-powerful programmer is rendered a mere player in an invisible game. Further, if programming offers a power that corrupts, what corrupts is not simply ease, but a certain combination of ease and difficulty, a certain experience of both power and frustration.[21] This programmatic combination of ease and difficulty creates, Weizenbaum argues, a new mental disorder: the compulsion to program, or hacking.

To explain this addiction, Weizenbaum again turns to gaming, drawing a parallel between obsessive programming and compulsive gambling – 'the magical world of the gambler' – since both entail megalomania and fantasies of omnipotence, as well as a 'pleasureless drive for reassurance'.[22] Like gambling, it is compulsive because it both rewards and challenges the programmer. It is driven by:

> two apparently opposing facts: first, he knows that he can make the computer do anything he wants it to do; and second, the computer constantly displays undeniable evidence of his failures to him. It reproaches him. There is no escaping this bind. The engineer can resign himself to the truth that there are some things he doesn't know. But the programmer moves in a world entirely of his own making. The computer challenges his power, not his knowledge.[23]

According to Weizenbaum, because the world of the computer takes place at the level of power rather than truth, it induces paranoid megalomania.[24] Because this knowledge is never enough, because a new bug always emerges, because an unforeseen wrinkle causes divergent unexpected behaviour, the hacker can never stop. Every error seems correctable; every error points to the hacker's lack of foresight; every error leads to another. Thus, unlike the 'useful programmer', who solves a problem at hand and carefully documents, the hacker is aimless: programming becomes a game without a goal and thus without an end. The hacker, that is, becomes like the computer – quickly and repeatedly reacting to inputs. Hackers' skills, Weizenbaum thus argues, are 'disembodied', and this disembodiment transforms their physical appearance: he describes them as 'bright young men of dishevelled appearance, often with sunken glowing eyes... sitting at computer consoles, their arms tensed and waiting to fire their fingers, already poised to strike, at the buttons and keys on which their attention seems to be as riveted as a gamer's on the rolling dice'.[25] Programmers as gamers as users: addicted to their screens, their bodies wasted by their habit. Their 'love' of programming, their contact with computers, produces wasted bodies.[26]

Although Weizenbaum is quick to pathologize hackers as pleasureless, pitiful creatures, hackers themselves emphasize programming as affectively pleasurable and their lack of 'usefulness' as what can actually be most productive and promising about programming. Programming, they insist, is not an addiction, but rather a healthy obsession. Linus Torvalds, for instance, argues that he, as an eternal grad student, decided to build the Linux core 'just for fun'.[27] Richard Stallman, who epitomizes Weizenbaum's description (and who drifted into the AI Lab as well) also emphasizes the pleasure, but more importantly the 'freedom' and 'freeness' associated with programming – something that stems from a conception of programming as other than the simple production of a commercial (or contained) product.[28]

What is important in fun is allowing oneself to go astray, or, more properly, the oscillation between fun as enjoyment and fun as obsession: an oscillation that produces the programmer as such. Also key is Weizenbaum's emphasis on games: as with the separation between fun and work, the distance between programmer and gamer seems to be disappearing. Thus, in order to understand fun and software, it seems crucial to understand the full embrace of games, as well as other forms of entertainment, within software culture.

Overenjoyment, or, in excess of fun

As Weizenbaum suggests, the common ground upon which programming and gaming both rest is the double valence of fun: fun as enjoyment and fun as obsession. In both cases, fun is experienced via an encounter with power and its limitations – both, that is, the limited sense of power that enjoyment brings with it and the larger sense of power, both sublime and sublimated, from under which one obsessively attempts to escape. The harnessing of fun's affective surplus perpetuates both this ultimate, conditioning power and the 'fun' it engenders. Thus, the classic example of the video-game obsessed programmer we alluded to before is not merely one in which he/she escapes from the task at hand into an entirely other game-world; instead, this game-world often recapitulates the same obsessive limitations that drive programming, but, again, with a different valence. Indeed, such a relationship between gaming and programming has in fact constituted a significant and extensive part of hacker culture. Long before the current concern with obsessive-compulsive Massively Multiplayer Online Gaming, back when online games were largely the concern of college students with access to the internet through shared, multi-user systems, dungeon adventure games like *NetHack*[29] encouraged, especially among computer science students, a conception of fun, as the experience of the limitations of one's power within a system that promises to enable one to constantly assert this power, not dissimilar to that which they would find in programming itself.

Despite, (or perhaps, indeed, partly due to) its limited graphical ambitions – the game is entirely represented in ASCII characters, and multiple attempts to update it with bitmapped graphics have proven largely unpopular –*NetHack* has proven to be *extremely* engaging on account of its absurdly totalizing ruleset. An open-source software project developed by a quasi-clandestine team of programmers, the game's fundamental assumption seems to be that every possible interaction within the game system should, under the right circumstances, be able to have a cumulative, interrelational effect. Thus, to offer but one well-known example, if, while blinded, and therefore groping your way through the dungeon, you feel your way around to the corpse of a cockatrice, you will be turned to stone when you touch it (unless, of course, you are also wearing gloves).[30] This proliferation of unexpected effects has given rise, in online discussion forums like the rec.games.roguelike.nethack newsgroup on Usenet, to discussions of 'Yet Another Stupid Death', and frequent proclamations that 'The DevTeam Thinks of Everything!'[31]

While it is a single-user game, the classic implementation of *NetHack*, originally released in 1987 (building on the heritage of an earlier game, *Rogue*, originally written in 1980) is in fact found on multi-user UNIX systems. Played in this way, the user/gamer is never entirely separated from the operating environment so that, for example, it is possible, when the system's user account receives mail delivered by the mailer-daemon (background process), for the user's character to be approached within the game by a 'mail daemon' (in-game monster) who then throws the character a scroll with the email message 'written' on it. Furthermore, the open-source nature (distributed under a modified version of the Bison general public license written by Weizenbaumian hacker

Richard Stallman) of the game encourages spelunking into the codebase as a way of trying to untangle the complex game dynamics (the *NetHack* Wiki, for instance, refers the reader of the Mail Daemon page to line 2773 of monst.c[32]) – something, of course, that is only possible if one is a literate programmer. The combination of operating system integration, open-source ethos (the official instruction manual is written by another well-known open-source advocate, Eric S. Raymond)[33] and the fantasy/role playing game setting all contribute to its niche popularity among engineers, computer scientists, mathematicians and other academic preserves of 'geek culture',[34] which is why it is not surprising that the *NetHack* homepage proudly displays the following quote from an anonymous Usenet poster: 'Thank you for the latest release of gradewrecker. My GPA just went in the corner and shot itself'.

NetHack, then, has often been an obsession (or addiction) through which programmers-in-training, even when taking a break from programming itself, learned to systematize, to 'think of everything', at the risk of their academic performance, and to have fun while doing so. Fun-as-enjoyment thus slides into fun as obsession in service of reinforcing the discipline, if not the practice itself, of software engineering. Indeed, as Phil Laplante and colleagues note, the programming challenges associated with developing the game's constantly changing, pseudo-random – and thus intensely engaging – structure have implications for developing similarly complex 'serious' systems for befuddling malicious attackers:

> Only a player who unlocks the secret 'wizard mode' and becomes omniscient is assured of winning NetHack. Likewise, a critical software application that incorporates deliberately misleading features, traps, and diversions could be the most reliably secure because only an all-knowing user could confidently use the application without self-destructing—in the sense, say, of the system crashing.[35]

The similarity here between text-based programming and text-based gaming is significant; in both cases, not only can the pleasure of the (ASCII) text easily turn to Weizenbaum's 'pleasureless drive for reassurance', but the hacker's obsession with coding-as-problem-solving that he relates to the gambler's urge can just as easily turn into an obsession with gaming-as-coding, the unpleasant need to further reassert one's power over the complex limitations of the game mechanics when one should 'really' be getting back to schoolwork, or something even more important, like one's social life.

Indeed, the connection between fun and compulsion can be located not only in the oscillation between programming and gaming as two increasingly similar modes of computing, but it can also take the form of an *affective* response, which, as we suggested earlier, is crucial to our interest in thinking through the relationship between software and obsession. Indeed, in autonomist Marxist thought, the distinction between labour and leisure time is under renewed pressure due to the rise of immaterial labour as a relation incorporating many activities and functions previously considered, in industrial economies, to be 'outside' the sphere of production. For Michael Hardt and Antonio Negri, for instance, immaterial labour is affective as well as cognitive, producing 'immaterial products, such as knowledge, information, communication, a relationship, or an emotional response'.[36] Furthermore, they argue, in an updated version of the classical Marxist base/superstructure distinction, that 'immaterial labor has become *hegemonic in qualitative terms* and has posed a tendency on other forms of labor and society itself',[37] informing even social practices that might still be understood as existing outside of this expanded productive sphere. Viewed this way, even 'leisure-time' activities that do not involve sitting in front of a computer subtly mirror and extend the affective affinities upon which labour in an information economy depends.

This is perhaps nowhere more apparent than in 1990s UK rave culture, a subculture that, not coincidentally, found one of its most welcome receptions Stateside in the San Francisco Bay Area, especially among the information workers of Silicon Valley and Multimedia Gulch.[38] Writing about rave music's relationship with the serotonin-releasing drug Ecstasy, Simon Reynolds describes 'the utopian/dystopian dialectic'[39] by which new sonic forms are generated as club culture itself moves cyclically from an optimistic engagement with the drug to something like soulless obsession. Describing a specific instance of the latter in late 1992, he writes:

> What I remember most of all is *the number of ravers whose smiles had been replaced by sour, cheated expressions*…the hollow numbness of veteran ravers whose brains have been emptied of serotonin, the 'joy-juice' which Ecstasy releases in a gush-and-rush of euphoria. Having caned E so hard for so long, these ravers find their pleasure-centre synapses are firing blanks.[40]

Crucially, Ecstasy is not considered to be physically addictive; yet, even here, an obsessive, mechanical dedication to the letter, the *logos*, of fun long after its spirit is gone, the willingness to keep on going whenever you get the feeling you've been cheated, is, again, precisely the flip side of fun, the point where it becomes so fun it's not. For Reynolds, this situation inevitably results in the euphoric re-birth of rave culture in another genre of electronic dance music, but what happens when a similar dialectic is played out at the level of the individual programmer/user? Or, to put it another way, what happens when the system being perpetuated is no longer a relatively benign subculture such as electronic dance music, but instead modes of production, systems of labour?

If, in rave culture, such a dialectic has produced new genres, in software engineering, as we have already begun to examine, it has produced instead higher levels of machinic abstraction. Building on the abstractions of automatic programming, Edsger Dijkstra's 1968 letter to the Association for Computing Machinery, 'Go To Statement Considered Harmful', as one of us has noted elsewhere, emblematizes a shift in the discipline of programming from a hierarchical model of control and punishment to a softer model of control, structured programming: 'Gotos make difficult the conflation of instruction with its product—the reduction of process to command—that grounds the emergence of software as a concrete entity and commodity'.[41] 'Good', that is, structured, programming allows this conflation at the same time as it also makes programming an art 'of organizing complexity, of mastering multitude and avoiding its bastard chaos'.[42] The 'good' comes from a move into abstraction and at the expense of the more 'direct' knowledge of automatic programming, itself already an abstraction in comparison to 'hard' coding. At a certain point, in other words, 'good' programmers, unlike Weizenbaum's hackers, stop obsessing about the best way to solve a particular problem, and instead relegate it to the domain of a standard implementation, relying, for example, on a generic function call instead of implementing the most efficient algorithm for each use case. Object-oriented programming, an even higher level of abstraction, extends this model further to incorporate code that the programmer has written him/herself.

Such abstraction not only effaces knowledge specific to 'lower' levels of coding, but moves the programmer away from the electrical basis of the machine and into a variety of structures for organizing complexity: development environments, source control systems and compilers, for example. In so doing, it further blurs the distinction between programmer and user. Writing programs at increasingly higher levels of abstraction, programmers become, in a sense, 'users' of their respective programming platforms rather than mythical masters of the machine.[43] Indeed, the similarities between what may at first seem to be three very different experiences of fun, that of the gamer, the drugged-up raver, and the programmer, become much more apparent when we consider that which specifically binds them together: the figure of the user. Such 'users', in enjoying the object of their use, whether drugs, games, or software development environments, are, at the same time, themselves *used*. Addicted to fun, they are pushed past enjoyment into obsession and back again in ever-new configurations nevertheless based on this principle.

If there is a radical potentiality in fun, then, it lies not in the simple act of gratification but, as we implied earlier, can only be generated through this dialectic between enjoyment and obsession. To give but one personal example, one of us had the – in retrospect, profoundly surreal – experience of not only raving but working at a software company in San Francisco's Multimedia Gulch in the mid-1990s. This period of employment began while said company was transitioning to a new, online-centric approach, making the atmosphere that of the rarest of mid-1990s tech companies: an internet company with an actual business model. Aside from the requisite free sodas, etc., provided, of course, for 'fun' (to foster, in other words, a round-the-clock work ethic), the one thing that sticks out about the experience is the red spiral slide, situated just next to the reception desk, which led to the floor below.

The slide was one of the first things that would have been pointed out to any visitor, prospective investor, or employee on a tour of the facilities, and people would always – *always* – be encouraged to try it out. 'Look, we have a slide!' became something of a cliché, both of the standard tour spiel (and a *Spiel*, of course, is a game, if you speak German), and – in the specific department (IT) where one of us worked, at least – as an over-enthusiastic, ideological fetish, a disavowal of marathon QA/bugfixing/maintenance sessions. The slide itself, of course, was more or less useless; most people who ended up sticking around for longer than just a tour inevitably tended to favour the stairs situated right next to it. What is most interesting about the slide, however, is that it was *worse than useless*. For the 'user', sliding down the segmented plastic always seemed to generate a buildup of static electricity, like rubbing your feet on a carpet – an irony we often joked about. Since the IT department, perpetually full – like any IT department – of half-disassembled hardware, was around the corner from the bottom of the slide, you always had to remember to touch a metal railing or doorknob before you went back to work, lest you inadvertently zapped a motherboard.

The whole experience of the slide, in other words, was something like a con, a hoax, one which blatantly encouraged an atmosphere of 'fun' as a means of extracting surplus value; you knew this even as you were sliding down it, trying nevertheless to enjoy it. But in so doing, it also generated an unintended and potentially entropic side-effect, quite aside from and extrinsic to either the subjective experience of enjoyment or the objective relations of production in which the slide was strategically situated. Crucially, the valence of this side-effect is *itself* indeterminate; it is ambi-valent. Long-term employees complained about *being* shocked by the slide as much, if not more, than expressing concern about the damage one could potentially cause to the high-tech environment. This chaotic surplus in the dialectic between enjoyment and obsession, this potentially destructive result of the law of unintended consequences, doesn't always, as in the case of static electricity, manifest itself physically, but can also – and perhaps most provocatively – flare up within the realm of software itself.

As our multifarious figure of the 'user' demonstrates across several seemingly different provinces of an informational economy predicated on affective, immaterial and 'free' labour, what is fun, what is engaging, is also often what is bad. This is, moreover, true of what is considered to be very bad programming. Paradoxically, 'bad programming', like *NetHack*, also has the potential to reinforce 'proper' ways of thinking about software development. There's a kind of fun in trying to ignore or subvert 'disciplined' practices of industrialized programming and a similar, obverse kind of fun in trying to figure out what a complicated and unstructured program is going to do without running it, but this 'fun', in both cases, *also* boils down to a kind of 'art', the art of obfuscated code. The International Obfuscated C Code Contest (IOCCC)[44] is perhaps the best-known example of such a practice. Twisting the valence on *NetHack*'s game-as-programming mentality further, obfuscated programming is itself a kind of game, in the sense that it is organized as a contest. Furthermore, many (though not all) of the submissions turn out, when properly compiled, to be games themselves. Entries like 2004's winner in the category of 'Best Calculated Risk', Brent Burley's draw poker program written in three lines of code (two of which are simply variable declarations), exemplify the sense in which the art of obfuscated programming both relies on a Dijkstran drive towards simplicity, and, at the same time, a complete inversion of the strictures of structured programming. In his entry, Burley writes, 'To keep things simple, I have avoided the C preprocessor and tricky statements such as "if", "for", "do", "while", "switch", and "goto"'.[45] Here, too, GOTO statements are considered harmful, not because they would make the program *more* difficult for humans to follow, but *less*.

Obfuscated programming is thus in some ways the obverse of the move towards a compartmentalized, structured programming that enabled the commodification of software. Instead of an 'abstraction [that] both empowers the programmer and insists on his/her ignorance',[46] obfuscated C, while it remains an abstraction (and is thus not somehow 'closer to the machine' for being un-/poorly structured) is one that disempowers (confusing/difficult code) while insisting on a greater degree of knowledge (hacks/tricks/obscure methods). For example, Carl Banks' 1998 *Best of Show* entry to the Obfuscated C Contest is written in the shape of an airplane, and indeed, when compiled, runs a basic X Windows flight simulator.[47] The code, however, only compiles with very specific switches – key mappings for controlling the plane, for example, must be specified at compile time – complicated enough that Banks specifically details them alongside his entry. While ignorance is related to discipline, and therefore empowering, here, useless knowledge is related to a kind of 'fun' which is disempowering.

Obfuscation itself is in fact the obverse of the drive towards rational communication. Etymologically, it has its roots in the Latin *obfuscare*, 'to darken';[48] it is the anti-Enlightenment. For Nick Montfort, obfuscated code, including the IOCCC and similar contests and traditions in the Perl language,

> darkens the usually 'clear box' of source code into something that is difficult to trace through and puzzle out, but by doing this, it makes code more enticing, inviting the attention and close reading of programmers. There is enjoyment in figuring out what an obfuscated program does that would not be found in longer, perfectly clear code that does the same thing…In this way it throws light on the nature of all source code, which is human-read and machine-interpreted, and can remind critics to look for different dimensions of meaning and multiple codings in all sorts of programs.[49]

As he points out, source code itself is located at a nexus between the human readable and machine executable, and obfuscated code thus highlights the potential for the play of meaning opened up by the disjuncture between these two different types of addressee. At the same time, however, obfuscated coding can be seen as a refusal of 'good programming' that also seems, paradoxically, to end up in its embrace. Consider, for example, as listed on the IOCCC homepage, some of the contradictory goals that the contest's organizers enumerate:

- To write the most Obscure/Obfuscated C program under the rules below.
- To show the importance of programming style, in an ironic way.

- To illustrate some of the subtleties of the C language.
- To provide a safe forum for poor C code. :-)

We might characterize these goals, respectively, as follows: to emphasize bad programming, positively; to emphasize good programming, negatively; to emphasize good programming, positively; and, finally, to emphasize bad programming, negatively (but humorously).

There is thus, again, precisely the danger we have seen in so many of the examples we have looked at thus far, that of fun folding back on itself as so much fun merely reinforces, consciously or unconsciously, positively or negatively, the 'right' way to do things. As our examples suggest, this vacillation between the positive and negative sides of fun, while not historically new (as the word's etymology itself indicates), has nevertheless proven exceedingly amenable to incorporation within a relatively contemporary system of political economy predicated on affective labour in general and increasingly reliant on 'free labour' in particular. The figure of the 'user' is emblematic of how this vacillation is encapsulated within such a system: programmers, gamers, ecstatic ravers – indeed, neoliberal subjects of all kinds – participate in the system by both using and being used *by* it. From Weizenbaum's hacker-gambler compulsively seeking mastery over the machine to the International Obfuscated C Code Contest, which, for 'fun', flagrantly subordinates the human to the computer, placing the programmer in the position of obsessively writing and reading software instructions deliberately oblique to human comprehension, the oscillating valences of fun have proven to be far from an obstacle to the perpetuation of this system. Indeed, they have thus far provided only the occasional glimmer, as in the spark of static electricity generated by the dot-com-era software company's slide, of anything that escapes its logic.

Yet there is one further goal that the contest specifically enumerates: 'To stress C compilers with unusual code', as Banks' flight simulator, with its eccentric compile-time options, does. This goal is in many ways precisely the most interesting because it seems to insist on relentlessly probing the gap between writing and execution that Montfort identifies as a key component of obfuscated code and which structured programming and other abstractions specifically seek to elide. But isn't this again, and perhaps ultimately, merely the obverse of the impulse to immediacy that fetishizing code as *logos* demands? Perhaps. Yet, and just as ultimately, it is here that the as yet unharnessed surplus generated on the battlefields of code and fun can be recuperated. Here, we would like to suggest, fun presents the possibility of neither a return to hardware nor a reified, fetishized software, but perhaps instead that eccentric, freakish and, perhaps most importantly, random or aleatory practices,[50] under the right circumstances, really *can* be subversive after all, if only on the most technical grounds.

These practices can be subversive, that is, as long as we gain enough knowledge from them that, instead of becoming 'better programmers', we become *disempowered* to the point where we can no longer accept the conflation of speech and action, the model of code as an instrumentalization of our thoughts. This would have the somewhat paradoxical effect of not only making our interfaces more productively spectral, as, again, one of us has suggested elsewhere,[51] but also revealing a site for action within the so-called immaterial labour of programming that is not reducible to *logos*. Writing about the relationship between another form of *logos*, juridical code, and violence, in relation to the question of the sovereign, Italian philosopher Giorgio Agamben suggests that '[t]he only truly political action…is that which severs the nexus between violence and law'.[52] The same could also be said of the nexus between execution and code. The image of a compiler choking on obfuscated C, then, holds within it the possibility that, in no longer substituting deed for word, we may also no longer automatically fall prey to that original sin of ideology itself: fighting phrases with phrases.

Notes

1 For example, Microsoft Office's *Ribbon Hero* series teaches knowledge workers about the more arcane features of the software giant's 'productivity' suite through the use of a game-like interface running on top of the suite itself. This funware is clearly modelled after the wildly popular *Guitar Hero* game series, in which the valence between 'fun' and 'work' is in fact only subtly different; whereas Office users can experience their 'work' as 'fun' through *Ribbon Hero*, *Guitar Hero* allows gamers to experience the 'fun' of playing music with only a fraction of the 'work' required to learn a musical instrument. See http://www.ribbonhero.com/ (accessed 12.12.2013).

2 See the analysis of 'free labour' in Terranova, Tiziana, *Network Culture: Politics for the Information Age* (London: Pluto Press, 2004), 73–97, discussed further below.

3 *Oxford English Dictionary*, online edition, s.v. 'fun', (accessed 04.03.2011).

4 Terranova, *Network Culture*, 74.

5 Ibid., 78.

6 We borrow this concept of 'bad in a good way' from Anna Watkins Fisher, who proposes in this phrase a rethinking of normative binaries through the figure of the parasite in her 'Feminist Impositions: Performing Parasites in Contemporary Art and Media' (PhD dissertation, Brown University, Providence, RI, USA).

7 *Oxford English Dictionary*, online edition, s.v. 'obsession', (accessed 06.06.2011).

8 As the figure of the 'user' provocatively suggests, fun is also bounded by addiction. Addiction, in Roman law, was the 'formal delivery of a person or property to an individual, typically in accordance with a judicial decision' (*Oxford English Dictionary*, online edition, s.v. 'addiction', (accessed 21.10.2011)). Addiction thus implies an unwillingly willing servitude to something.

9 Ahmed, Sara, *The Cultural Politics of Emotion* (New York: Routledge, 2004), 10.

10 For Ahmed, building on Marx's theory of circulation, objects around which an affect intensifies merely carry emotions, the true currency of affective economies: 'affect does not reside positively in the sign or commodity, but is produced as an effect of its circulation' (ibid., 45), much like the distinction between classes can be perceived through the uneven circulation of money (and, going further, its transformation into capital) in the Marxian model. Unlike Terranova, for whom, coming out of the Italian school of autonomist Marxism, affect is specifically tied up with questions of political economy surrounding immaterial labour in general and free labour in particular, Ahmed sees affective economies as existing separately from economies of capital. Instead, she writes, 'What I am offering is a theory of passion not as the drive to accumulate (whether it be value, power or meaning) but as that which is accumulated over time' (ibid.). Yet in software culture, the experience of fun, an emotion which Ahmed does not explicitly consider, can be seen to coincide with both of these conceptions of affective economy; as fun accumulates over time, it both generates its affective subject, the programmer/user oscillating between enjoyment and obsession, *and* bolsters capital, as that which is behind the 'funning', in its drive to accumulation. Indeed, Ahmed cites Marx's own alignment of affect with economy: 'This boundless drive for enrichment, this passionate chase after value, is common to the capitalist and the miser' (Marx, Karl, *Capital: A Critique of Political Economy, Volume One*, trans. Ben Fowkes (Harmondsworth: Penguin, 1976), 254, cited in Ahmed, *The Cultural Politics of Emotion*, 45). For our part, we might adapt Marx's phrasing here to say that this boundless drive for enjoyment, this passionate chase after fun, is common to the programmer and the user.

11 Chun, Wendy Hui Kyong, *Programmed Visions: Software and Memory* (Cambridge, MA: MIT Press, 2011).

12 See Backus, John, 'Programming in America in the 1950s—Some Personal Impressions', in *A History of Computing in the Twentieth Century*, (eds) Nicholas Metropolis, Jack Howlett and Gian-Carlo Rota (New York: Academic Press, 1980), 127.

13 Guttag, John V., 'Abstract Data Types, Then and Now' (1977), in *Software Pioneers: Contributions to Software Engineering*, (eds) Manfred Broy and Ernst Denert (Berlin: Springer, 2002), 445.

14 Weizenbaum, Joseph, *Computer Power and Human Reason: From Judgment to Calculation* (San Francisco: W. H. Freeman and Company, 1976), 115.

15 Foucault, Michel, *The Birth of Biopolitics: Lectures at the Collège de France 1978–1979,* ed. Michel Senellart, trans. Graham Burchell (London: Palgrave, 2004), 120. See also Lison, Andrew, '1968 and the Future of Information', in *The Global Sixties in Sound and Vision: Media, Counterculture, Revolt*, Timothy Scott Brown and Andrew Lison (eds) (New York and Basingstoke: Palgrave Macmillan, 2014), 245–74.

16 John 1:1, *The Bible, King James Version*.

17 See Jacques Derrida's analysis of *The Phaedrus* in 'Plato's Pharmacy', in *Dissemination*, trans. Barbara Johnson (Chicago: University of Chicago, 1981), 134.

18 Brooks, Fredrick P., *The Mythical Man-Month: Essays on Software Engineering* (1975), (New York: Addison-Wesley Professional, 1995, anniversary edition), 8.

19 Weizenbaum, *Computer Power and Human Reason*, 277.

20 Ibid., 234.

21 Ibid., 115.

22 Ibid., 121–2, 124.

23 Ibid., 119.

24 For more on the relationship between paranoia and knowledge, rather than truth, see Chun, Wendy Hui Kyong, *Control and Freedom: Power and Paranoia in the Age of Fiber Optics* (Cambridge, MA: MIT Press, 2006).

25 Weizenbaum, *Computer Power and Human Reason*, 116.

26 For more on the relationship between emotions and bodies, see Ahmed, *The Cultural Politics of Emotion*, 4.

27 Torvalds, Linus, *Just for Fun: The Story of an Accidental Revolutionary* (New York: Harper Collins, 2001).

28 Hacking reveals the extent to which source code can become a fetish: something endless that always leads us pleasurably, as well as anxiously, astray. For more on this, see Chun, *Programmed Visions*.

29 See http://www.nethack.org/ (accessed 12.12.2013).

30 For this and the many other possibilities that may arise in the game from interacting with cockatrices both living and dead, see David Corbett's spoiler page archived at http://alt.org/nethack/mirror/www.geocities.com/dcorbett42/nethack/cockatrice.htm (accessed 12.12.2013).

31 See http://nethackwiki.com/wiki/Yet_Another_Stupid_Death and http://nethackwiki.com/wiki/The_DevTeam_Thinks_Of_Everything (both accessed 12.12.2013).

32 See http://nethackwiki.com/wiki/Mail_daemon and http://nethackwiki.com/wiki/Scroll_of_mail (accessed 19.03.2014).

33 Raymond is in many ways an emblematic link between open-source culture and the conception of fun we are exploring here. Aside from authoring the *NetHack* instruction manual, he maintains the often humorous 'Jargon File' of hacker slang (see http://www.catb.org/jargon), a print version of which has been published as Raymond, Eric S., ed., *The New Hacker's Dictionary,* (Cambridge, MA: MIT Press, 1996, 3rd edn) and has also played a significant role in the revival of the 'weird' programming language INTERCAL. See Raymond's INTERCAL site at http://www.catb.org/~esr/intercal/ and also Mateas, Michael and Nick Montfort, 'A Box, Darkly: Obfuscation, Weird Languages and Code Aesthetics' in *Proceedings of the 6th Digital Arts and Culture Conference*, IT University of Copenhagen (December 1–3, 2005), 144–53 (available online at http://nickm.com/cis/a_box_darkly.pdf; both URLs accessed 12.12.2013), a revised version of which, focusing on 'Weird Languages', can be found as Mateas, Michael, 'Weird Languages' in *Software Studies: A Lexicon,* ed. Matthew Fuller (Cambridge, MA: MIT Press, 2008) 267–75. As Mateas and Montfort suggest, 'weird languages' themselves share many similarities with obfuscated code, discussed below.

34 See especially http://nethackwiki.com/wiki/NetHack#Why_do_people_like_NetHack.3F (accessed 12.12.2013).

35 Laplante, Phil, Hurlburt, George, Miller, Keith and Voas, Jeffrey, 'Certainty through Uncertainty?', *Computer* 44(2) (Feb. 2011): 80. While the use of 'wizard mode', included in the game for debugging purposes, is not, strictly speaking, cheating, it is nevertheless considered an illegitimate method of winning by any serious *NetHack* player; see http://nethackwiki.com/wiki/Cheating (accessed 12.12.2013).

36 Hardt, Michael and Antonio Negri, *Multitude: War and Democracy in the Age of Empire* (London: Penguin, 2004), 108. See also Lazzarato, Maurizio, 'Immaterial Labor', trans. Paul Colilli and Ed Emory in *Radical Thought in Italy: A Potential Politics*, (eds) Paolo Virno and Michael Hardt (Minneapolis: University of Minnesota Press, 1996), 133–47.

37 Hardt and Negri, *Multitude,* 109.

38 For more on the relationship between rave culture, autonomist Marxism and globalization, see Lison, Andrew, 'Postmodern Protest? Minimal Techno and Multitude', in *Between the Avant-Garde and the Everyday: Subversive Politics in Europe from 1957 to the Present*, Timothy S. Brown and Lorena Anton (eds) (New York: Berghahn Books, 2011), 201–18 and 'Love's Unlimited Orchestra: Overcoming Left Melancholy via Dubstep and Microhouse', *New Formations* 75 (Summer 2012): 122–39.

39 Reynolds, Simon, *Energy Flash: A Journey through Rave Music and Dance Culture* (London: Picador, 2008, updated 20th anniversary edition), xxvi. See also Reynolds' chapter on the rave scene in the San Francisco Bay Area, which illuminates the links between drug counterculture, electronic music subculture and Silicon Valley technoculture, ibid., 287–94.

40 Ibid., 191. Emphasis added.

41 Chun, *Programmed Visions*, 36.

42 Dijkstra, Edsger W., 'Go To Statement Considered Harmful', in *Software Pioneers: Contributions to Software Engineering*, (eds) Manfred Broy and Ernst Denert (Berlin: Springer, 2002), 352. The connection here between the necessity of structured programming for 'mastering multitude' and Hardt and Negri's thesis that it is 'the set of all the exploited and the subjugated…the multitude, whose struggles have produced Empire', the system of global capitalism predicated on immaterial, affective labour, 'as an inversion of its own image', is particularly evocative (Hardt, Michael and Negri, Antonio, *Empire* (Cambridge, MA: Harvard University Press, 2000), 393–4). Additionally, for Mateas and Montfort, '"Good" code simultaneously specifies a mechanical process and *talks about* this mechanical process to a human reader' (Mateas and Montfort, 'A Box, Darkly').

43 For a different, but by no means incompatible perspective, see Kittler, Friedrich, 'There Is No Software', *ctheory.net*, 18 October 1995, http://www.ctheory.net/articles.aspx?id=74 (accessed 12.12.2013). Kittler argues that computing culture's emphasis on software obscures the fundamental nature of all computing in its present form as voltage manipulation.

44 See http://www.ioccc.org/ (accessed 12.12.2013).

45 See http://www.ioccc.org/2004/burley.hint. The code itself can be found at http://www.ioccc.org/2004/burley.c (both accessed 12.12.2013).

46 Chun, *Programmed Visions*, 37.

47 See http://www.ioccc.org/1998/banks.hint. The code itself can be found at http://www.ioccc.org/1998/banks.c (both accessed 12.12.2013).

48 *Oxford English Dictionary*, online edition, s.v. 'obfuscate' (accessed 28.03.2011).

49 Montfort, Nick, 'Obfuscated Code' in Fuller, *Software Studies*. Focusing on obfuscated code, Montfort's piece is, like Mateas's 'Weird Languages' in the same

volume, a revised and edited version of Mateas and Montfort, 'A Box, Darkly' (see note 33).

50 See the introduction to this book.

51 Chun, *Programmed Visions*, 21.

52 Agamben, Giorgio, *State of Exception* (2003) trans. Kevin Attell (Chicago: University of Chicago Press, 2005), 88.

Monopoly and the Logic of Sensation in *Spacewar!*

Christian Ulrik Andersen

In his book *Homo Ludens* (written in 1938), the play theorist Johan Huizinga states that the instinct for play, and 'having fun' that defines its essence, is core to human civilization. Play is not merely a cultural manifestation; it rather fulfils a general necessity. Civilization is played: it 'arises in and as play, and never leaves it', he claims.[1] Play is what constitutes our culture, and, as Huizinga describes, it can be found in children's games as well as poetry, music, law and warfare. However, playing has a double nature. Play entices a civilized form (as when the judge wears a wig), while, at the same time, playfulness consistently tries to challenge or even destroy that form. As pointed out by another play theorist, Brian Sutton-Smith, play is carnivalesque (as when someone else wears the judge's wig).[2] So, even though play is widely accepted as a cultural and civilized activity that gives meaning to meaningless actions, and builds institutions and industries that perpetuate this meaning, it simultaneously attempts to outgrow its civilized form, to displace meaning and to disrupt its institutions.

The game of *Spacewar!,* developed by a group of 'hackers' at MIT in the early 1960s and popular in a wide circle of programmers, marks the beginning of a history of computation produced and performed for fun. In what ways does *Spacewar!* displace and disrupt the formation of meaning and the institution within which it appears?

At the time, technology was conventionally understood as an enterprise dealing with mechanics, but as Norbert Wiener expresses it, the 'new industrial revolution' perceives the machine 'not as a source of power, but as a source of control and a source of communication'.[3] With computers, the ability to control information became embedded in technology, in the tools and models that simulate the world, and are able to predict and control future events.

Rather than augmenting the physical power of the human, technology began 'augmenting human intellect', according to Douglas Engelbart.[4] Cybernetics and the study of information feedback systems which exhibited a potent usefulness for a wide range of areas, including biology, psychology and economics, originated in Radiation Lab's research on aircraft defence at MIT during World War II. Computer science of the 1960s, with MIT at the heart of it, continued the development of computerized, 'smart' control of airspace.

Despite a shift to the augmentation of the mind, the role of the body is central to *Spacewar!* Unlike the military air defence systems' air control, which operates rather differently, the game of air control in *Spacewar!* is a thrilling experience where the software is designed to challenge the player's bodily control of the computer – literally, letting the player push buttons in order to control a spacecraft on a screen and experiment with the augmentation of the human body. What does this playful reinvigoration of the body in an information control system impose? And which understanding of humans, software and its institutions does playing and making such games imply?

The history of making games as political and disruptive acts is highly informative in this respect. In particular, the popular game *Monopoly* presents an exciting case study. While it is not widely known, *Monopoly* was intended to be pedagogical and provide anti-capitalist education about the negative consequences of the United States' monopolization of land in the early twentieth century. This paradox of an anti-capitalist game, which to many incarnates a celebration of capitalism, reveals a bodily relation between play and meaning that can help build an understanding of the computer scientists bodily play with computers six decades later. Not being able to determine the precise meaning of gameplay can indicate that the kind of sense-making that takes place is closer to the joyous affirmation of a Nietzschean 'play of the world', rather than to finding a particular intended signification.[5]

The Landlord's Game: A politics on land

In 1904 Elizabeth J. Magie, a Quaker woman from the United States, patented the board game *The Landlord's Game*. As explained by Burton H. Wolfe in *The Monopolization of Monopoly*, Magie was a follower of Henry George's theory that private monopolies on land and the renting of land produce an increase in the value of land which profits a few landlords rather than the majority of

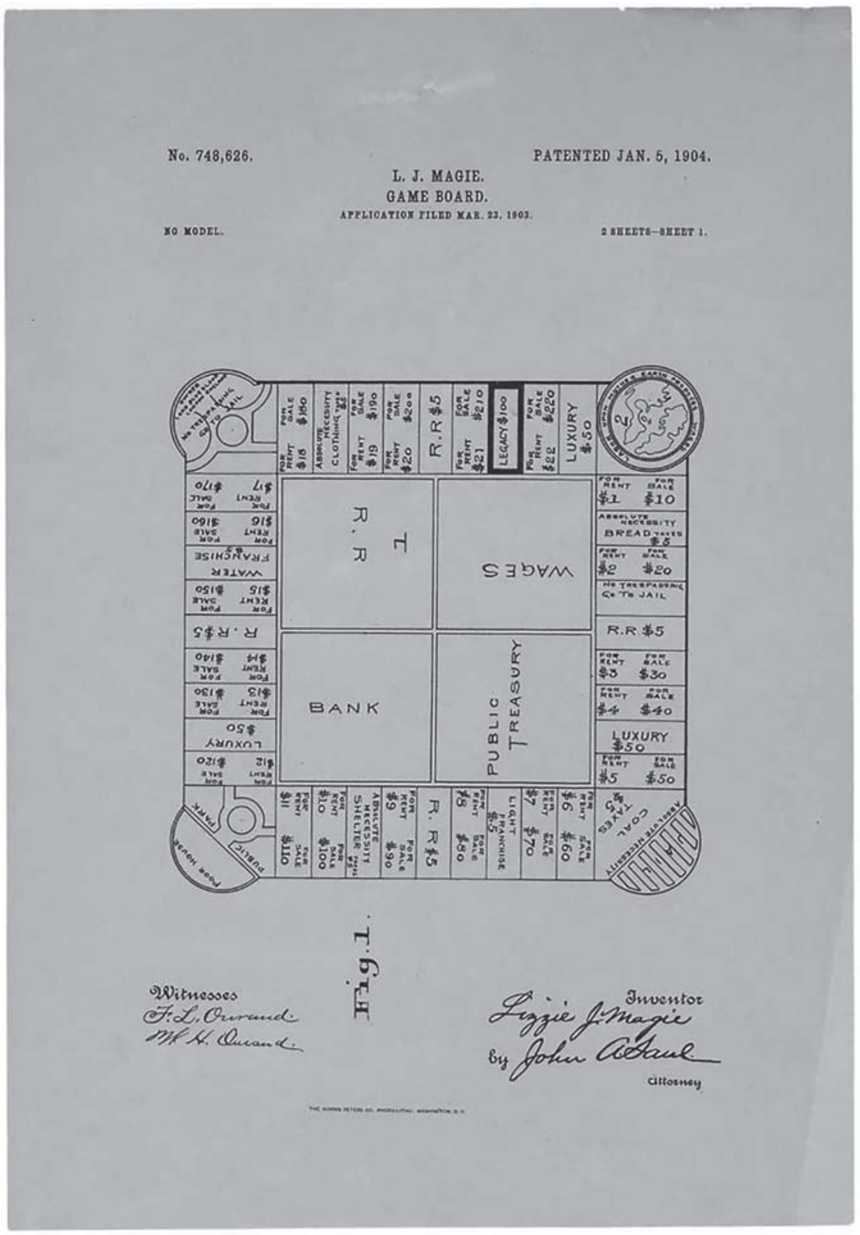


Figure 9.1 Printed patent drawing for a board game invented by Elizabeth J. Magie (*The Landlord's Game*). The game was designed to illustrate the principle of a single-tax idea, proposed by Henry George for use in the United States. In 1935, Magie sold her patent to Parker Brothers, the publishers of *Monopoly*

Source Record of the Patent and Trademark Office, National Archives, United States tenants.[6] Henry George proposed a 'single tax' on land to discourage speculation and balance the relationship between owners and tenants. In *The Landlord's Game*, players are allowed to explore the processes of monopolization with the deliberate intention of educating them in Georgism. Playing a game is about the tactile exploration of and adaptation to the codes of conduct in the game, and Magie's political intention was, in short, to create a practical, educational demonstration of the negative consequences of private monopolies on land.[7]

Though many similar games were played at the time, *The Landlord's Game* is considered the main inspiration for *Monopoly*, the world's best-selling board game.[8] In other words, modelling the process of the monopolization of land seems to exhibit a purely procedural logic that does not entail any consecutive conclusion: the same game may be perceived as either a celebration or a critique of capitalism and the monopolization of land. What does this say about the relationship between the game and its meaning?

The fact that *The Landlord's Game* had the potential to turn into a cheerful celebration of capitalism first of all exemplifies the ambiguous nature of play. Play serves two seemingly opposing goals: it reproduces prevailing values but it also diverts unacceptable impulses and drives them into personally and socially acceptable activities. Play also has a 'carnivalesque' and 'frivolous' nature, as Brian Sutton-Smith puts it.[9] Under the circumstances of play, you are allowed to assume other roles (e.g. a greedy capitalist) and act out unacceptable behaviours (e.g. brutality). This means that the demonstration of monopolization does not necessarily lead to reflection and awareness; it may also be linked to frivolity and the opportunity to experience the thrill and 'fun' of being an insatiable capitalist in a socially acceptable way.

The frivolity and fun of playing may lead to joyful immersion in greed, but the pedagogical scope of the game is not just about building arguments pro et contra capitalism. In *The Landlord's Game* there is something frivolous in the very acts of naming and creating rules such as: 'Poverty Place' (land rent $50), 'Easy Street' (land rent $100) and 'Lord Blueblood's Estate' ('no trespassing – go to jail').[10] As a demonstration of the monopolization of land, the game can be seen as a playful act of comparing politics to board games. However, to grasp the critical nature of *The Landlord's Game*, one must employ a different explanatory framework and explore what the fun and frivolity of this kind of demonstration and designation imply.

In the context of a discussion on the nature of humour (in *Logique du sens*), Gilles Deleuze sheds light on this problem. Deleuze discusses Diogenes' ability to argue by using demonstrations, and praises his ability to show and designate.[11] Deleuze's inclusion of Diogenes is presumably an implicit reference to Søren Kierkegaard. In Kierkegaard's thinking, irony is a path from aesthetics to ethics, whereas humour is the path to a religious being. Deleuze does not quote Kierkegaard's notion of humour directly and thus avoids the religious aspect of his philosophy. However, he refers to his notion of irony.[12] Kierkegaard draws upon Diogenes' method of argumentation and, as an example, presents his argument about the ambiguity of movement. Diogenes' opponent assumed that movement could only be described in terms of instances, but in his response Diogenes rises and walks back and forth.[13] Plato laughed at people who were satisfied with being shown and showing examples, as Diogenes did at those who did not ask *who* was but *what* was, etc.[14] But Deleuze supports Diogenes' way of thinking and points to his ability to tear down Plato's idealism and essentialism by humorously replacing ideas with examples. In other words, meaning and signification are destroyed by designation (and humour), and it is pointless to inquire into the signification of the demonstration. According to Deleuze, the substitution, designation and manifestation (introduced below) that take place in humour, which destroy relations to meaning and signification, also differ from both Socratic and romantic irony. The latter seeks the 'real' signification (the true meaning), and the former remains caught in an indeterminacy of signification ('does the ironic enunciation mean one thing or another?'). Games are, as demonstrations of processes, ruled by designation. The meaning of the game, whether it is to be capitalist or anti-capitalist, is arbitrary, as it is a demonstration.

We may thus conclude that the critical aspect is not intrinsic to the game; it must lie elsewhere. In another section of *Logique du sens* Deleuze draws a semantic distinction between signification, designation and manifestation. A signification entails a connection between a word and a concept; a designation between a sentence and external circumstances; and a manifestation between a sentence and a speaking subject.[15] If *The Landlord's Game* is to be considered political, it should not only be viewed as the *demonstration* of a procedural logic; it must also be seen as a *manifestation* of an alternative. Naming streets, creating rules and creating a parodic and humorous demonstration of capitalism must be understood as a political manifestation of an anti-capitalist movement, and not just as a demonstration of the dynamics of capitalism. It is only if seen through this lens that its challenge to capitalism becomes unmistakable.

As the game is a manifestation it should be examined in relation to the ones who manifest themselves through the activity of play. In other words, to regard *The Landlord's Game* as political and critical, one must not only attend to what is played and how it is played, but also to *who* is playing the game. The critical players are not the consumers of *Monopoly* (enjoying the fun of playing large-handed capitalists), but the Georgists and Quakers making the game. One of the early players of *The Landlord's* Game wrote:

> those who wanted copies of the board for *Monopoly* took a piece of linen cloth and copied it in crayon. It was considered a point of honor not to sell it to a commercial manufacturer, since it had been worked out by a group of single taxers who were anxious to defeat the capitalist system.[16]

The Landlord's Game was not a consumerist game but a folk game developed and modified by the players themselves. Ultimately, 'getting the message' depends on the players' ability to identify with the project. One could even argue that there is no political and critical movement unless there are places where such performances and manifestations can occur. This performative aspect seems to be a core characteristic of political and critical games, whose activity otherwise remains ambiguous and whose meaning arbitrary.[17]

Focusing on the performative action of the player is important to our understanding of what it means to play with software, and what it meant to play *Spacewar!* in the 1960s. In fact, one could argue that the very concept of using computers for gaming and play depends on the manifestation of a critical attitude towards the institutions where computers are applied, and the kind of logic that lies behind them: playing games challenges how cybernetics governs not only the user but also life itself.

Spacewar!: A politics of the senses

In the original version, *Spacewar!* is a game for two people, each of whom controls a spacecraft in a two-dimensional world. The object of the game is to shoot the opponent with missiles while manoeuvring and avoiding the gravity well of the 'star' at the centre of the screen. *Spacewar!* was first conceived in 1962 at MIT. Although usually attributed to Steve Russell, it was a collaborative project of 'geeks' or 'hackers', as they were also labelled at the time, which indicates that the computer game was made by and for computer researchers who shared and sometimes contributed to the game. For example, one player/programmer, Peter Samson, was offended by the 'random stars' in the background and added a program to control the background void based on real star charts which, referring to the price of computers at the time, was entitled *Expensive Planetarium*. Another player/programmer, Dan Edwards, introduced gravity into the game and thus significantly improved the gameplay.[18]

Taking up the theme of the space war in the midst of the Cold War cannot be considered random. *Spacewar!* was developed only a year after Yuri Gagarin's first space travel. Essentially the game itself must be understood as a play on Cold War themes, a satire in which the race for space is turned into popular culture's play: the spacecrafts of the Cold War are replaced by spacecrafts visually resembling those of Tintin (in *Objectif Lune*) and *The First Lensman* (a popular science fiction novel by 'Doc' Smith); and the computer interaction of military defence systems is turned into play with the cinematic model work of beams and explosions.[19]

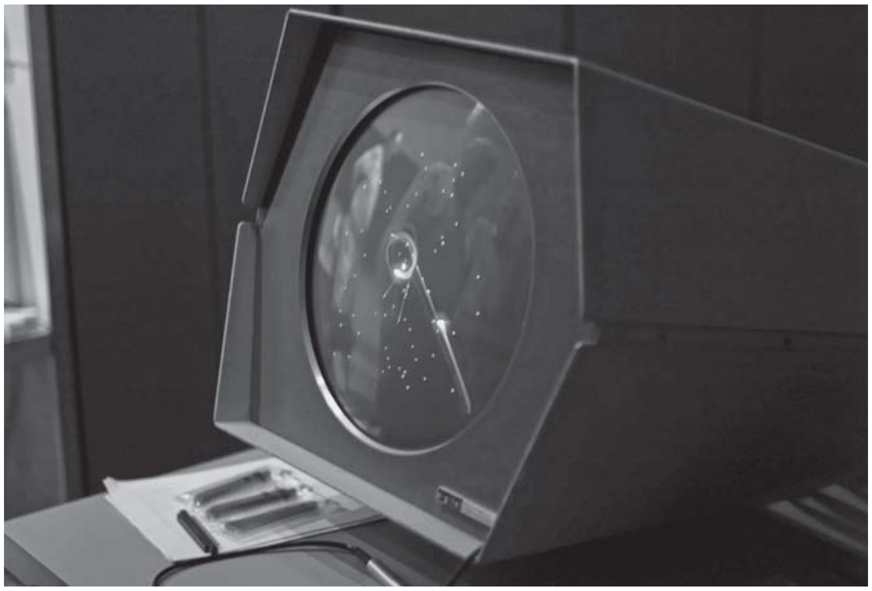


Figure 9.2 *Spacewar!* running on PDP-1. Duelling players fired at each other´s spaceships and used early versions of joysticks to manipulate away from the central gravitational force of a sun as well as from the enemy ship

Source Image by Joi Ito

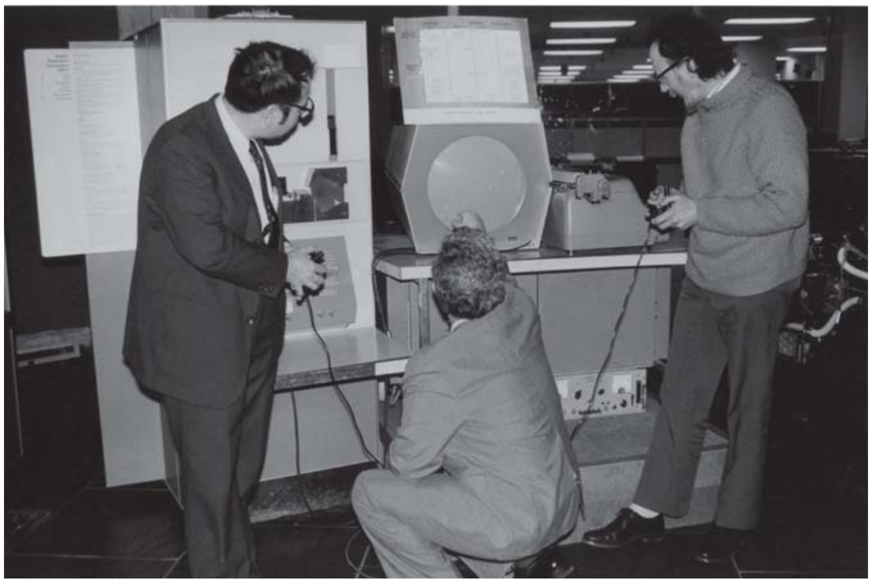


Figure 9.3 Alan Kotok, Steve Russell, Martin Graetz play *Spacewar!* 1983 ca.

Source Image courtesy of Computer History Museum

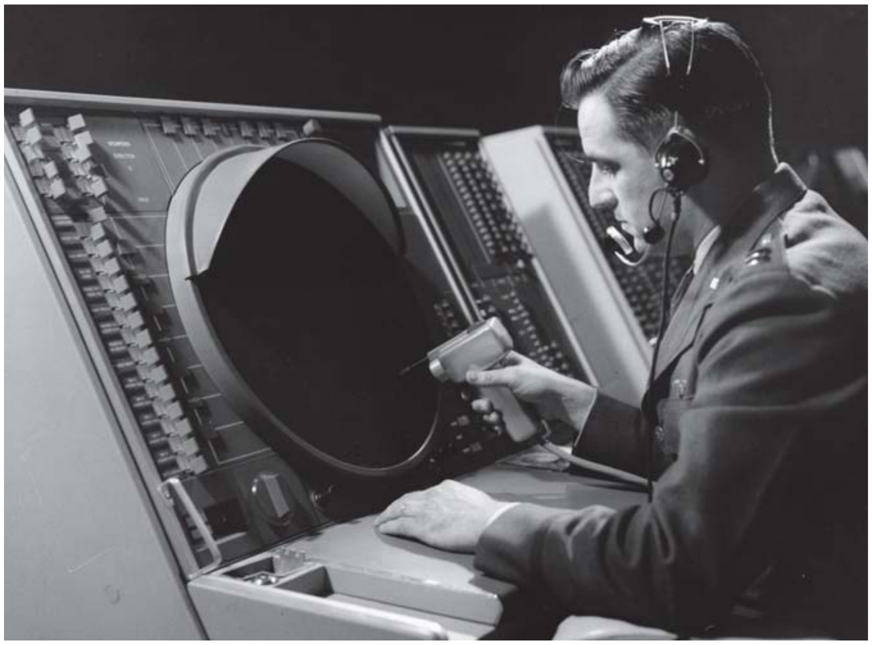


Figure 9.4 Operators of SAGE used light guns to pinpoint aircraft tracks. When a blip appears on the scope, the light beam causes the computer to assign a track number and to relay speed, direction and altitude information to various consoles. Publicity photograph of a SAGE console operator with 'gun' pointed at the screen. 1959 ca.

Source Image courtesy of Computer History Museum, and used by permission of the MITRE Corporation

The gameplay echoes one of the most common Cold War computer systems, SAGE. During World War II, the bomber aircraft gained increasing importance in warfare. The time between spotting enemy flights and the required response at high speed was constantly diminishing, even given the use of radar. The ability to predict flight tracks and target positions was crucial for air defence, especially with the development of the nuclear bomb. In the early 1950s, the US military joined forces with researchers at MIT and IBM to produce a Semi Automated Ground Environment system (SAGE). On the basis of a computerized cybernetic system and a number of physical control posts, it became possible to predict flight tracks and automate the interception.[20] The whole gameplay of *Spacewar!* is essentially an imitation of the operation of SAGE. In front of a radar-like display the player's task is to predict the tracks of missiles and intercept the enemy spacecraft. The frivolous nature of play adds a hyperbolic and parodic dimension to the imitation: in *Spacewar!,* the player can control the aircrafts in the outer hemisphere, shoot enemies down and experience the thrill of explosions.

As a *demonstration*, *Spacewar!* is science fiction and does not seem to model a complex process, or even a process that remotely imitates reality (unlike *The Landlord's Game*). The game is science fiction, as whether the Cold War would lead to real space wars was unforeseeable at the time. Hence, the game is not political – it does not seek to make significatory claims about the world. So, what does the game demonstrate and what kind of politics is at play?

As a *manifestation*, the game is based not on similarity but on substitution. In *Spacewar!* the Cold War is replaced by play and battles from popular culture. This manifestation is humorous in Deleuze's sense of the term. Replacing SAGE with sci-fi explosions is absurd and defies signification, but, paradoxically, the thrill of shooting down an enemy spacecraft while performing the parody also makes sense. On one level, of course, it makes sense because of the internal logic of the game, where shooting missiles at the opponent leads to victory. First and foremost, however, it makes sense because it is thrilling and fun: it makes sense as a bodily sensation. The game is a manifestation of the body as a sense-making machine; it simultaneously makes sense (bodily) and remains non-sense. In Deleuzian terms, 'the logic of sense' becomes 'the logic of sensation'.

In this logic of sensation one must not only pay attention to what takes place *in* the game (illustrations, demonstrations, etc.), but also to the body of the player and the sensation of playing. In fact, one might argue that the bodily engagement, 'feeling' the gravity, the thrill of overcoming obstacles and even the shouting and excitement of playing are central experiences to all computer games. Playing computer games is an uncanny experience of the body's simultaneous (entrapped) presence and absence in a cybernetic system.

Playing a computer game is an experience of the body as an indispensable element that regulates the cybernetic feedback loops in the game. In *Spacewar!* the player constantly seeks to predict the trajectory of the spacecraft and its missiles on the screen, as whenever the spacecraft steers towards the centre of gravity the player needs to counter steer.

However, playing the game on the screen is simultaneously accompanied by an experience of bodily absence: there is no body to make sense with in the cybernetic system of the game-world. There is no 'feeling' of gravity, steering and speed, but only the 'illusion' of a body. Today, experiencing a computer game is compulsory in popular culture and does not seem very revolutionary, but in 1962, to playfully foreground the status of the body as a mediatory and sense-making element (both present and relied upon and absent and repressed) in a cybernetic feedback system was exactly what made *Spacewar!* a rebellion. Playing the game reflected the human and bodily experiences called upon and repressed by SAGE in particular and military computer science in general.

SAGE depends on human users watching the screen, tracking aircraft manually and reporting to the system. Humans make the defence system run but also represent a potential disturbance: they may miss their target or choose not to act. In response to human fallability, cybernetics quantifies the human by reducing him/her to an element that can be calculated. The objectification and quantification of the human is core to a certain version of cybernetics and was addressed by Norbert Wiener, who, in his last book, asks: 'Can God play a significant game with his own creature? Can any creator, even a limited one, play a significant game with his own creature?'[21] Here, Wiener points to the technocratic and administrative nature of cybernetic systems: they are control systems that seek to control the future. In this, they assume the position of a 'God' and thus mark the end of free will and the bourgeois subject: the human is no longer in control of the system, but has become an element in the system. In cybernetic systems, humans enter into a 'symbiosis' with computer systems, as J. C. R. Licklider, one of the key developers of SAGE and a well-known theorist in the field of cybernetics, called it.[22] Media theorist Brian Holmes notes that the human becomes an 'info mechanic being' whose 'double constitution could be felt in the uncanny identity of the strange new creatures that fired the guns and piloted the planes: both seemed to waver between machine-like, implacable humans and intelligent, humanlike machines.'[23]

This shift in the construction of the subject and subjectivity was certainly not limited to computer systems. Cybernetic thinking provided useful conceptual, epistemological and ontological frameworks to economics, management, sociology, communications engineering, behavioural psychology, HCI and other design branches, among others, where cybernetic ideas are either foundational or remain partially unchallenged. One example is the use of game theory to analyse war strategies. Based on the theoretical game 'prisoner's dilemma', John von Neumann (a key figure in both the development of the nuclear bomb and the computer) advocated for a preventive war against the USSR. In 'prisoner's dilemma' two prisoners, who together have committed a crime, must decide individually whether they should acknowledge the crime and betray the other. Only the one that acknowledges the crime and betrays the other will go free, but if they both acknowledge their crime, they will both be sentenced. A third option is to keep quiet, in which case they both get a lesser sentence for a lesser offence. Most likely, the prisoners will betray each other. As an allegory on the possible war between the United States and USSR, the debate was whether one should 'keep quiet' or 'wipe the other out'.[24] The idea of a preventive war was (luckily) rejected by the Truman administration, but game theory still played a role in the justification of the Cold War arms race. [25] In the application of game theory to politics, which at the time also laid the ground for the actual computerized defence systems, the human and, not least, the human body have been reduced to estimations of potential casualties in a system.

The symbiosis of human and computer emerges as the reality of the post-war subject, and the reality of the subjectivity of the computer scientist. The development of computer systems has always been linked to military strategic objectives such as ballistics or cryptography. In the United States, during the Vietnam War, it was even proposed that financial support for computer research should be dependent on its strategic and military impact.[26] In other words, at the time of *Spacewar!,* computer research was also military research, undergirded by cybernetics.

Computer researchers (including the makers of *Spacewar!*) were involved in a political cybernetic system which produced strategic military cybernetic systems (such as SAGE). In their daily work life, they were creating the systems to which they themselves were quantified objects, subjects with an 'uncanny identity between cruel, machine-like humans and intelligent, human-like machines'. Employing the computer as a creative tool, giving programmers as well as users agency and 'free will' of parody, making games to which to respond bodily, playing and being frivolous must be seen as a rebellion against the quantification and objectification of the body and of the human. If *Spacewar!* reveals anything about the system dynamics and procedural logic of the Cold War, it is in particular the impact of political power on the bodies and lives of its creators and players. In defiance of their situation, the casualties of cybernetics, the human programmers stage the performance of a cybernetic symbiosis between a human and a machine body as a playful and joyous act.

Playing with software: The redistribution of the senses

Play reproduces behaviours (as rituals), but its frivolous nature also makes it capable of producing change. From time to time, people create new games that challenge how we sense and make sense of the world, and the mechanisms that control these processes. To adopt the words of Jacques Rancière, play may have a similar political role to that of aesthetics in the 'distribution of the senses'.[27] *Spacewar!* is not political – it does not represent an overt struggle between the institution and 'the hackers' (the programmers/players). Hacking was commonly accepted; but the hackers were striving to speak for themselves through play. As information workers at MIT they were part of an established order of cybernetic information systems, but as humans they were subjected to the systems they were creating. Through the aesthetic practice of making games, playing and having fun with software, they could be seen as reclaiming their right to sense and make sense of the world in a non-quantified and non-objectified way, defying the new available forms of being, knowing and expressing oneself.

To grasp the revolutionary aspects of *Spacewar!*, one must not only consider what the game demonstrates (the space war) but also what it is a manifestation of. *Spacewar!* is revolutionary not because the activity of shooting spacecrafts demonstrates any inherent procedural and persuasive rhetoric that opposes the political or social realm, but because it is a manifestation produced by a group of people, which challenges cybernetics and the way it is embedded in the sophisticated apparati that control and represent the senses (such as the Cold War, its defence systems and even the computer labs themselves). Unlike *The Landlord's Game, Spacewar!* does not try to model or oppose processes that even remotely resemble reality. Rather, the game is part of a movement that turns Cold War computer science into a game in which players/programmers gain an experience of what the cybernetic apparatus otherwise seizes control of: the body as sense-making.

What did this 'programmer identity' and reclaimed right to sense and make sense of the world through play lead to? The game became an icon of a culture of programmers who valued free inquiry, knowledge sharing and a distaste for authority. Stewart Brand describes this culture in a 1972 issue of *Rolling Stone* magazine, featuring a young Alan Kay at Xerox PARC, who said that 'The game of *Spacewar!* blossoms spontaneously wherever there is a graphics display connected to a computer'.[28] It is, at least partially, through playing with software that the players/programmers assumed and spread this ethos. New visions of how computers could serve people as tools for sharing knowledge and creativity were created.

The computer as a means to bodily sensation eventually also led to a whole new industry: the video and computer game industry. *Spacewar!* was the first widely used software program.[29] The popularity of *Spacewar!* is an early indication of the potential of the game industry and a foreseer of computers being brought from the labs to the masses through coin-operated video game arcades in the 1970s. In fact, as early as 1971 a variation of *Spacewar!* entitled *Computer Space* was used in one of the first attempts to commercialize computer games (by Nolan Bushnell, who made the ever-popular *Pong* in 1972 and who later founded Atari).

The fate of computer games and these new visions of software is another story. Ironically, the US military probably learned more from *Spacewar!* than from SAGE. The military has a long tradition of using games to train soldiers, and computer games have provided new possibilities for combat training, among other roles. For example, the game *America's Army*, produced by the US Army and available for free on their website, which gives players a sense of 'the real deal' of warfare, is also used as a branding mechanism to help recruit soldiers. And vice versa: as the gaming industry has developed, the makers of computer games have learned from surveillance systems such as SAGE. In particular in massive, multiplayer online games, player activity is seen as a key mechanism in the production of value. An increase in the number of social relations players engage in during the game, or the time players spend in the game, raises the potential value of the platform for advertisement. With the capitalization of not only the games but also the activity of play itself, the player continues being objectified, quantified and entrapped in the cybernetic system.

Through our playful acts new institutions and regimes of power and control have arisen, including the regime of play and games itself. Revolutionary play with software in the 1960s soon went hand in hand with new marketing strategies that made computer games accessible to everyone. The 'gamification' of other types of software, using the principles of play to make software an engaging experience, is a central strategy to arenas such as social media. Can play in this environment continue to challenge our institutions and hierarchies of meaning and control? Johan Huizinga provides an easy answer: 'animals have not waited for man to teach them their playing', he claims.[30] Play is a natural thing: cats play with mice, dogs pretend to bite while playing, some insects engage in foreplay before mating and so on. Our animal instinct for play cannot be repressed. We simply need to have fun to live. In Michel de Certeau's terms, playing may have lost as a strategy, but it persists as tactics.

Notes

1 Huizinga, Johan, *Homo Ludens: A Study of the Play Element in Culture* (London: Routledge, 1980), 173.

2 Sutton-Smith, Brian, *The Ambiguity of Play* (Cambridge, MA: Harvard University Press, 1997), 11.

3 Wiener, Norbert, 'Men, Machines, and the World About', in *The New Media Reader*, N. Wardrip-Fruin and N. Montfort (eds) (Cambridge, MA: MIT Press, 2003), 71.

4 Engelbart, Douglas, 'Augmenting Human Intellect – A Conceptual Framework', in *The New Media Reader*, N. Wardrip-Fruin and N. Montfort (eds) (Cambridge, MA: MIT Press, 2003), 95–108.

5 Jacques Derrida suggests free play, and 'the affirmation of the play of the world' as the de-construction of significations, rigid concepts and binary oppositions rooted in *logos* and prevailing Western metaphysical thought: 'a Nietzschean *affirmation*, that is the joyous affirmation of the play of the world and of the innocence of becoming, the affirmation of the world of signs without fault, without truth, and without origin which is offered to an active interpretation'. Derrida, Jacques, 'Structure, Sign, and Play in the Discourse of the Human Sciences', in *Writing and Difference*, ed. Jacques Derrida (London: Routledge, 1998), 292.

6 Wolfe, Burton H., 'The Monopolization of Monopoly' (2011), excerpts from *The San Francisco Bay Guardian* (23 April 1976), http://www.adena.com/adena/mo/index.htm (accessed 12.12.2013).

7 The game may be seen to have what game researcher Ian Bogost calls a 'procedural rhetoric', which simply means that the game makes a claim about how something works by modelling its processes. Bogost, Ian, *Persuasive Games – the Expressive Power of Video Games* (Cambridge, MA: MIT Press, 2007).

8 The two games are almost identical. The only exception is that players in *The Landlord's Game* do not own but rent land, and the object of *The Landlord's Game* is not monopolization but educational. As stated in the introduction to the game: 'The object of this game [*The Landlord's Game*] is not only to afford amusement to players, but to illustrate to them how, under the present or prevailing system to land tenure, the landlord has an advantage over other enterprisers, and also how the single tax would discourage speculation'. Quoted in Wolfe, 'The Monopolization of Monopoly'.

9 Sutton-Smith, *The Ambiguity of Play*.

10 Wolfe, 'The Monopolization of Monopoly'.

11 Deleuze, Gilles. 'Dix-neuvième série de l'humour', in *Logique du sens* (Paris: Les Éditions de Minuit, 1969), 159.

12 Ibid., 165.

13 Kierkegaard, Søren. 'Gjentagelsen', in *Samlede værker*, vol. 5, ed. S. Kierkegaard (Copenhagen: Gyldendal, 1991), 115.

14 Deleuze, Gilles. *Logique du sens*, 160.

15 Ibid., 22–4.

16 Wolfe, 'The Monopolization of Monopoly'.

17 Many contemporary political and critical computer games, most of which exist on the internet, have inherited the ambiguous nature of *The Landlord's Game*. For instance, the game *Super Columbine Massacre RPG!* (2005) may be interpreted as a demonstration of the dynamics behind the high-school killings, but may also be seen as a new documentary genre. Responses to the game are contradictory, and there is no intrinsic mechanism in the game to encourage the latter interpretation, unless one considers the developer Danny Ledonne's position as an American documentary film director who was bullied as a child in school, as a priori ethical. Jose Antonio Vargas, 'Shock, Anger over Columbine Video Game – Designer Says Web Creation an "Indictment" of Society', *Washington Post* (Saturday, 20 May 2006).

18 J. M. Graetz, 'The Origin of Spacewar', *Creative Computing* 7(8) (1981), http://www.wheels.org/spacewar/creative/SpacewarOrigin.html (accessed 12.12.2013).

19 The inspiration from popular culture is also documented by one of the creators of *Spacewar!,* Graetz, ibid.

20 The system was fully operational up till 1983, and the overall idea persists. In 1983, Ronald Reagan introduced the SDI (the Strategic Defense Initiative that included space-based systems for air defence), which was later to be followed by

other missile defence systems. The SDI was perceived as science fiction by some and was labelled Reagan's 'Star Wars' project.

21 Wiener, Norbert, *God and Golem, Inc*. (Cambridge, MA: MIT Press, 1964).

22 Licklider, J. C. R.. 'Man-computer Symbiosis', in *The New Media Reader*, N. Wardrip-Fruin and N. Montfort (eds) (Cambridge, MA: MIT Press, 2003), 74.

23 Holmes, Brian, 'Future Map or How the Cyborgs Learned to Stop Worrying and Love Surveillance', *The Laboratory Planet* 63(2) (2008): 4–7.

24 Poundstone, William, *Prisoner's Dilemma: John Von Neumann, Game Theory and the Puzzle of the Bomb* (New York: Doubleday, 1992).

25 To continue this line of argument, the arms race can be explained as analogous to the 'dollar auction'. A dollar is put up for auction, but unlike other auctions, any participant in this auction will have to pay his/her highest bid, no matter if he/she wins the auction or not. The most successful strategy is to keep bidding, even if the bid exceeds $1. If winning the auction symbolizes victory in a possible war, the theory suggests that no one will dare to withdraw from the arm race, even though it far exceeds the prize to be claimed. Ibid., 262.

26 Gere, Charlie, *Digital Culture* (London: Reaktion Books, 2002), 130.

27 Rancière, Jacques, *Le Partage du sensible* (Paris: La Fabrique Éditions, 2000), 14.

28 Brand, Stewart, 'SPACEWAR – Fanatic Life and Symbolic Death among the Computer Bums', *Rolling Stone* magazine (7 December 1972): 50–8.

29 The code for *Spacewar!* was distributed for free by Digital Equipment Corporation, the leading vendor of computers in the 1960s.

30 Huizinga, *Homo Ludens*, 1.

10

Human-computer Interaction, a Sci-fi Discipline?

Brigitte Kaltenbacher

He's not a man or a machine: A Terminator, a cybernetic organism.
Kyle Reese, *Terminator,* 1984

It's life, Captain, but not life as we know it.
Spock, *Star Trek,* 1963

It isn't faith that makes good science, Mr. Klaatu, it's curiosity.
Prof. Barnhardt, *The Day The Earth Stood Still*, 1953

History of HCI, cybernetics and *Terminator*

James Cameron shot his iconic film *Terminator* (now *Terminator 1*) in Los Angeles in 1984. An impressive creation of cybernetic fiction, the Terminator, a Frankenstein-like humanoid robot imbued with artificial life, is a powerful figure offering a futuristic vision of technology, which neatly separates power to affect and power to be affected between technology and humans. The Terminator can pass for a human on an instrumental level as he displays natural body movement, language use and a goal-directed behaviour. The cyborg's social conduct, however, is delivered with the rudimentary capacity of a reptilian brain, only capable of fight or flight responses and a 100 per cent focus on control and survival. Throughout the movie the Terminator's face remains a motionless blank. The film plays with the immediacy of cinematic real-time footage and the hypermediacy[1] of CGI-enhanced and superimposed game interfaces, anticipating today's mix of passive and active attention media. The Terminator is the perfect metaphor of how HCI models the human at that time: an enticing yet dated fiction, drawing us in by seductive presentation.

In the same year, 1984, the first British HCI group called 'Interaction' was established by the BCS (British Chartered Institute for IT), two years after the first conference devoted to HCI was held in Gaithersburg, United States, by the SIGCHI (Specialist Interest Group Computer-Human Interaction), a subgroup of the Association for Computing Machinery. Initially a mix of human factors (HF), ergonomics, behavioural science and computer engineering, the newly formed discipline was united[2] by agreeing on the use of scientific methods. These design methods for human efficiency in the workplace derived from Taylorism and the ideas of scientific management.[3] I agree with Dourish when he comments on this approach to HCI as being based entirely on philosophy of the pre-1930. Computer science in practice involves reducing high level behaviours to low-level, mechanical explanations, formalizing them through pure specific rationality; in this, computer science reveals its history as a part of positivist, reductionist tradition.[4]

HCI's traditional academic home was computer science. However, cognitive science was also an important contributor to the newly formed discipline, and a contributor that suffered too from a rigorous Cartesian separation of mind and matter, due to its cybernetic roots. At the Macey Conference 1941 McCulloch stated that 'brains do not secrete thought as the liver secretes bile, but … they compute thoughts the way computing machines calculate numbers'.[5] This idea culminated in HCI's human processor model,[6] which likens the human brain to a computer processor. The model brings the fictitious concept of the purely rational, goal-driven and linearly progressing human in alignment with the computational automation theories, and creates the user through the application of the theories of AI. Crucially, it is also the first wave of cybernetics' idea of technical communication as a feedback loop between tracking, control and prediction that still fundamentally informs HCI's understanding of the human-machine relation.

Another misconception of the discipline becomes apparent in the historical and, at times, contemporary synonymous use of the acronym 'HCI' in textbooks and publications both for 'human-computer interaction' and 'human-computer interface',[7] which points to the HCI's reductive tendency to flatten the complexities of interaction processes into interface design. The space between users, systems and their representations in interfaces is brimming with layers of technology, mediation and organizational structures, yet HCI's literal translation of control into (interface) control tools still manages to collapse this multifaceted space into its surface.

In the early 1990s Cameron and the Terminator moved on, yet HCI did not do the same in relation to the human user. In *Terminator 2* the original cyborg is faced with an improved, next generation model – fluid, highly adaptive, technically improved and even more seductive and deceptive on an 'interface' level: Cyborg v2 can shape shift and so create a multitude of fictitious humans. Yet, it is ultimately beaten by its older, technically inferior version. Why does this happen? The main reason is that Cyborg v1 learned to tune into human emotions[8] and to reach into the social realm, a leap that the HCI discipline had – and has – yet to achieve. The 1990s mainstream HCI still focused on usability,[9] which could be engineered, modelled and optimized by experts, even without any user involvement. The lowest level of user-centred design (UCD) only requires to imagine user's input.[10]

1994 marked the departure from a dominance of the single user-computer interaction and a shift towards ubiquitous, distributed and networked computing, due to the internet becoming available to the public. The production of new cultures of networked computer users that occurred as a consequence was not necessarily obvious to the discipline of HCI. As Booth pointed out in his *An Introduction to Human-Computer Interaction*, soft sciences such as '[s] ocial psychology and sociology have been classed as the *neglected disciplines* as they are not given the representation and coverage they deserve within the literature, given the importance of the problems they might address'.[11] Moreover, HCI's designs for fictitious users seen through a scientific lens were leaving the real users with a variety of emotions ranging from puzzlement to frustration and despair. Most HCI teaching material at the time reiterated the need to produce 'less off-putting solutions',[12] with Booth voicing an urgent 'need to overcome hostility towards users'.[13] The infamous 'Clippit'[14] perhaps sums up the idea of fun and user friendliness in HCI design of the 1990s; probably the most unpopular (and most ridiculed) software feature ever.[15] Perhaps that is why at that time Alan Cooper[16] started developing a specific rather than generalized understanding of users. He viewed the concepts of an imaginary 'user' and 'user-friendliness' as too vague and argued that such concepts invited designing for stakeholders or according to the producers' wish lists of features, which were then projected onto 'the elastic user'.[17] His 'personas',[18] on the other hand, were supposed to represent a variety of user types within targeted market segments, which were differentiated according to demographics, attitudes and/or social status. Alas, personas were still fictional characters, i.e. discrete multiplicities akin to the T1000 rather than qualitatively enriched considerations of humans, and since this method has become standard practice in digital interaction processes, designing for fictitious rather than real users continues to live on.

However, also in 1995, Don Norman coined the term 'user experience' to expand HCI's limited concept of usability. With this concept, Norman intended to address all aspects of user experience with the system,[19] taking HCI beyond interface interaction. Perhaps unwillingly, he started an assault on the discipline's self-proclaimed scientific, objective and rationalistic stance that continues to haunt it. Since its popularization, Norman's term has drawn attention to users' subjective, affective and emotional perceptions in digital or technical interaction. While this helps enrich the role of the human in the technical interaction equation, dealing with it has proved to be hugely challenging both to traditional and aspiring HCI and user experience (UX) practitioners, and remains so.

HCI, the user experience and fun

Almost two decades after the term emerged, the idea of UX is still in flux. An understanding of HCI as the single user-interface interaction has been displaced by the distributed, embedded or even invisible interactions supported by compu ting devices, which communicate with us – and for us – via interconnected networks. In turn, digital media industry has formed its own institutional bodies to deal with the UX, such as: the UxDA (User Experience Designers Associations Group) or IxDA (Interaction Design Association), which include respective communities, conferences and practices. At the same time, such groups, initiatives and emerging fields keep revisiting, exploring and modifying established interaction practices and HCI literature, thus maintaining the prevalence of reductive tendencies. Even popular academic writing on 'emotional design' and 'affective computing' reinstates the concept of the mind as an information processor:[20] in Picard's *Affective Computing,*[21] emotions improve efficiency in decision-making and in Norman's *Emotional Design*,[22] they make us react favourably to attractive interface designs – but both authors categorically conceptualize the usability parameter of satisfaction as the binaries of loving or hating, positive or negative affects, the digital 'on' or 'off'. Perhaps that is why Garret, an industry practitioner and author of the highly successful

The Elements of User Experience, claims that 'any user experience effort aims to improve efficiency'.[23] However, to say that UX should be fun or pleasant can be just as reductive and binary as to say that the usability of a digital product should be satisfactory. An ISO (International Organization for Standardization) standard on user experience was formulated[24] in 2009, but it is still not clear what is meant by the term.[25] The concepts of fun, enjoyment and pleasure, which are used interchangeably to describe it, are elusive.[26] Nonetheless, traditional and new interaction design communities are largely in agreement that UX should be fun, enjoyable and pleasurable.[27]

A recent trend in HCI and UX design is to view the user experience through gaming: heightened attention, intense engagement and the persuasiveness of the genre are precisely those desired UX qualities that digital interaction design aspires to achieve. Gaming is also seen as a route through which a more complex understanding of fun and pleasure can be constructed. However, this, in my opinion, promising approach early on suffered from an unhelpful oversimplification now referred to as 'gamification'[28] – a mechanical view of games as rule-based systems that advance through the interplay of tasks, user reactions, feedback and rewards, e.g. points, awards or scores. Gamification as such a simplified abstraction of game mechanisms to create engaging user experience outside games has also been criticized by game designers and researchers as 'not being fun and creating an artificial sense of achievement',[29] displacing complex game mechanics with simple reward schemes,[30] and reducing the emergent properties of the games experience to a rule-based system with measurable goal achievement and frequent feedback.

Ferrara's 'Playful Experiences, Creating Game Experiences in Everyday Interfaces' claims to look beyond gamification when investigating games to inform UX design. He views this attempt as quite natural, since 'both UX and game design are forms of human-computer interaction, [which] inherently share some common theory, objectives, and practices'. [31] Then, despite stating that fun has an emergent nature, Ferrara models the 'playful experience' directly onto Garrett's (already mentioned) efficiency oriented UX model, with only a slightly adjusted vocabulary. He suggests that following the planes of the model in UX design will have 'enjoyment and fun emerge from the experience. … Conversely, fun dies when any of the planes haven't been adequately addressed',[32] effectively reducing fun to a predictable property after all.

It seems that HCI's cybernetic heritage, and the paradigm of tracking, control and prediction as the only communication model continues to persist:

a pleasant experience is part of a deterministic system – it functions as a predictable response to a correctly designed stimulus. Fun, thus, becomes the fashionable update on 'flashes' or 'blinks',[33] i.e. interface features that catch users' eyes in the race for a prime slot in attention economy, where any competitor is only ever a mouse click away.[34] Because of this continued reductive tendencies of the practice, progressive HCI/UX thinkers argue that it is time for a 'third paradigm of HCI'[35] to emerge, which will embrace an integral approach to understanding the user experience such as embodied interaction and user's situated meaning. Considering such intangible and emergent aspects could help understand the human technology interaction experience in the contemporary socio-technical environment in greater depth and, in turn, help engage with it.

Exploring the UX through a deeper look into game design and research, in my opinion, is particularly well suited not only to help us understand the powerful and complex world of emotions in relation to this context, but to mark a watershed moment for the HCI and UX design community. The departure from an understanding of emotions solely as reactions to interaction encounters in favour of their productive potential plays a crucial role in this paradigm shift. Going beyond a simplistic understanding of the qualitative aspects of the UX, such as emotional feedback, it overcomes the paradigms of measurement and predictability in digital interaction practice, and problematizes residual deterministic and reductionist thought models, while working towards a fluid emergent and evolving framework.

In the next section I will explore the various concepts of emotion to arrive at a multifaceted understanding of experience. It is informed by a cross-reading of popular HCI literature, game and interaction studies and cognitive psychology, especially educational psychology and learning sciences. A focus on the latter allows me to discuss a user group that has been ignored so far both by traditional and progressive interaction design communities: competently co-producing 'new media-literate' and creative digital participants. This discussion could help extend the current interchangeable and nondescript use of 'fun, pleasure and enjoyment' in UX practice into a wider-ranging 'experience' vocabulary as well as establish them as distinct markers outlining a rich experience landscape. Similarly, a critical reading of Csikszentmihalyi's concept of *flow*[36] in human engagement, utilizing neuroscientific accounts of human attention, attempts to debunk the fictitious conception that derives from the appropriation of *flow* by UX design, which suggests that the components of *flow* could act as design parameters.

Towards a richer understanding of fun and engagement

In their article 'The Semantics of Fun' Marc Hassenzahl and Mark Blythe[37] tackle a few issues that current UX and interaction design discussions grapple with: the indiscriminate use of fun, enjoyment and pleasure in describing user experience, the lack of structured theories to apply these terms more explicitly and the need to deploy these concepts competently to design for engaging interaction experiences. By introducing fun and pleasure as polarities, with many shades between them, they help us think about user experience and complexity of emotions: here, 'fun' is aligned with distraction and 'pleasure' with absorption. Investigating their scale from a psychological, political and contextual perspective, the authors develop connotations such as 'triviality, repetition, [and] spectacle' for fun and 'relevance, aesthetics, [and] commitment' for pleasure. While pleasure seems to benefit from more substance, and fun seems to stand for the fleeting enjoyment and volatile experiential aspects, there is no judgement on the quality of experience, as in 'engaged is good and distraction is bad'.[38] The authors point out that this scale merely caters for the varying needs of users, i.e. sometimes to be engaged and sometimes to be distracted. In addition, they touch on the derogatory dimension of fun, when it is used to belittle, as well as its political and subversive potential: fun can be used as a distraction in the workplace, where it 'can be seen … as a resistance to the rigid demarcation between work and leisure'.[39] In the context of media industries, fun has become a commodity that can be bought and consumed.

Pleasure, contrarily, is seen as a more fulfilling and deep form of enjoyment. Here Blythe and Hassenzahl's discussion overcomes a potentially reductionist stance on several accounts. First, they add a contextual dimension. Secondly, aesthetic pleasures in their work account for shared cultural and social values of experience (as opposed to the execution of the interface). Thirdly, the concepts of progression and commitment that they use seem particularly well suited to distinguish user experience from the usability factor 'satisfaction', rather than setting them as somewhat synonymous. Commitment as part of the pleasant experience helps overcome challenges in interaction, a view that is echoed in game studies: Lazzaro, for example, distinguishes between 'hard fun' and 'soft fun'.[40] 'Hard fun [is] the overcoming of obstacles [and] compelling challenges that demand sophisticated strategies', while variants of 'soft fun' include the engaging, enjoyable and the social aspects of sharing a game.[41] Next, Hassenzahl and Blythe turn to progression, for them an essential aspect of experience: it is a generative concept that 'stimulates [and] surprises'. 'Surprise marks the central difference between satisfaction and pleasure. Satisfaction is the emotional consequence of confirmed expectations, whereas pleasure is the consequence of deviations from expectations'.[42] Discussing surprise also marks a significant departure from predication. Certainly, not all surprises are pleasant, yet 'unexpectedness' has the exciting potential to open new directions and generate new perspectives. Pine and Gilmore, authors of the industry classic *The Experience Economy*, agree that surprises are key to transforming mundane encounters into memorable experiences.[43] Throughout their discussion, Hassenzahl and Blythe stress the contextualization of experiences as imperative, and that it is impossible to design experiences; one can only design for them.[44]

While their discussion is helpful when attempting a multifaceted take on fun and pleasure as part of the user experience, the route they choose to illustrate the psychological dimension of engagement comes across as uncritical. They refer to Csikszentmihalyi's concept of *flow*[45] as one of the few psychological accounts of pleasure, which is surprising for two reasons: first, *flow* marks a state of intensity, a mode of attention and, thus, the emotion of pleasure may be an after-effect of flow, but does not have to be,[46] and secondly, more up-to-date and better researched work that has been published on the subject since 1990,[47] such as neurophysiological accounts of the interaction between human behaviour and pleasure is not mentioned or used. Thus, *flow* is essentially a state of continued high alertness, which is triggered by the release of neurotransmitters. To be exact, 'dopamine "fixes" attention, makes thinking more efficient, and also mediates feelings of elation and pleasure'.[48] While the concept of the *flow* can act as a metaphor for an intensely engaged experience, and, as we will see later, promotes the importance of this state for learning and creativity, reductive tendencies can creep in when looking at the components of *flow* as a checklist or design guide for the 'optimal experience': a close match between skill and challenge, clear goals and constant feedback to control the performance.[49] Consequently a reduction of *flow* to a set of conditions to be satisfied revives the simplistic models of experience HCI has long been suffering from.

Yet, while *flow* might not be the all encompassing answer to designing a pleasant UX, its connection to intense experiences can also be linked to pleasure's darker side: obsession and addiction. Video games present a good example. Neuro-imaging studies of video gamers showed that rapid shifts between different salient perceptions increased the release of dopamine during game activities[50] resulting in the feeling of pleasure; MRI scans during (simple)[51] video games showed intense activity in brain regions associated with addiction.[52] Such intense experiences are prone to backlashes in flow producing burnout or 'gamers' regret'.[53] Hence exploring all aspects of intensity in users' experiences, beyond the equilibrium of superficial pleasantness, leads to a multifaceted understanding of emotions as part of users' experience in interaction, and a richer notion of engagement. Indeed, it is this potential for the intense experience the game industry successfully taps into that makes game interaction a cultural and commercial benchmark for contemporary interaction design.

Fun and games; from HCI's fictitious cyborg to the complexity of creativity and co-creating users

As already discussed, it is tempting to view games mechanically as rule-based – reward systems. But once the focus shifts to the quality of such rules as simple, open and therefore variable and flexible, their full potential for intense human engagement unfolds. Tom Chatfield's investigation of fun in video games forms an important part of the research on the emotional complexity and intensity in the gaming experience. Chatfield starts by discussing functional gaming aspects and concepts dear to HCI and UX designs' heart: rule-based systems, simple interactions and frequent feedback, with added 'visceral thrills' for excitement. The novelty lies in the ways in which he ties them in with their expansive antagonists. Chatfield notes insightfully: 'we are rule-loving creatures and this love extends into play',[54] but essentially 'emergent behaviours are central to video game theory',[55] as game play is closely related to discovery, experimentation and open-ended exploration.[56] Similarly, simple interactions are not linearly repeated facilitators of 'ease of use',[57] but are central to emergent behaviour. Visceral feedback (e.g. jaw dropping sounds and attractive images), albeit fun, cannot drive or sustain engagement, because such repetitive short-lived 'kicks' quickly flip pleasure into exhaustion. Instead of interpreting 'reality' as realistic representations of perceptions, Chatfield points out that the most successful games are realistic in terms of the range of social interactions they offer.[58] Ultimately, he argues that pleasure in the context of gaming is closely intertwined with meaningful (inter)action beyond instant gratification or skill optimization: '[l]earning (…) is perhaps the most vital [human] trait in evolutionary terms'.[59] Video games can feed this evolutionary need by acting as 'learning engines' for the kind of learning that occurs in the context of game playing: active, iterative and experiential learning in a non-functional context with time set aside.[60]

Recent publications in learning sciences, a branch of cognitive science that focuses on education, support Chatfield's assertions about video games' potential to facilitate active and experiential learning, their strong social and evolutionary standing, and the importance of non-functional context and the expanded time frame. Guy Claxton, a professor at the learning science department in Bristol, offers an explanation for the relationship between evolutionary history and a non-functional context: complex organisms, such as fish, can detect, register and make use of patterns, such as rock formations, to avoid the dangers they might face at low tide. The next step up is moving from passive pattern reception to active exploration through curiosity: rats as well as monkeys are proactive when it comes to interaction with their environment. 'Being receptive, attentive and experimental' are evolutionary functions built into the brain, and unless there are more pressing issues at hand, no further encouragement is needed.[61] This means that once fed, rested and, when the survival is secured, complex organisms, including humans, will start to explore, experiment and play, circumstances and time permitting.

The neuroscientist Antonio Damasio shares Claxton's perspective both in terms of his evolutionary approach and in perceiving emotions as active drivers of our behaviours (and not as simple reactions to stimuli). Damasio distinguishes between emotions and feelings in order to discuss the complex interplay between our non-conscious and conscious affects. Emotions are automatic actions of the body, some of which literally set the body into motion (as in the above-mentioned exploratory excursions). They range in scale from basic, in the case of reflexes, to, in complex organisms, high level emotions, such as joy, fear, shame and other, and spark into our consciousness as feelings, in particular to indicate needs or point at change. Indeed, Damasio's widely accepted 'somatic marker hypothesis' illustrates crucial impact of emotions on cognitive processes in efficient decision-making.[62] Moreover, sole emotional reasoning 'can promote outcomes [i.e. actions] that could have been derived rationally'.[63] In other words, non-conscious decision-making can lead to action that is perfectly reasonable, but derives from (the much faster) affective rather than conscious cognition. Lastly, we not only think, but also learn, unconsciously. Such implicit learning (i.e. intuitive and unstructured), Claxton notes, can be superior to explicit learning (structured and rational learning) when it comes to dealing with complex patterns of contingency, such as puzzles, the Rubrik's cube or gaming.[64] Sometimes implicit learning is the precursor to conscious knowledge; at other times it produces tacit knowledge that stays in the unconscious.[65] Either way, these explorations by neural and learning sciences establish cognition and learning as deeply intertwined conscious and non-conscious processes, and experiential learning as an active process, driven by the inquisitive trait of our emotions.

It is now clear that the concept of the *flow* connects to learning in a complex and fundamental way: in moments of intense focus our rapid shifts of attention not only oscillate between conscious perception and cognition, but also their non-conscious 'shadows', as Damasio calls them. Hence, the concept of the *flow* works on a metaphorical level to describe an intensely engaged experience, yet its pleasure or the effects of experiential learning it produces cannot be reduced to the successful feedback mechanism that closes the loop of self-organizing skill optimization in autotelic, i.e. especially suitable, personalities. Instead, pleasure in this context opens up to include active exploration and learning of the new and to creativity. The metaphor and concept of the *flow* is helpful when it is connected to a distorted sense of time that defies the Newtonian stopwatch. It would exceed the scope of this chapter to discuss time from a qualitative perspective in terms of the experienced time (where time flies or takes ages), but such time is, nevertheless, an important condition of creative experience. Time pressure, on the other hand, where efficiency becomes synonymous with 'as fast as possible', channels the brain into the familiarity of repetition, where experimentation, exploratory learning, creativity and fun cease.

Active, iterative and experiential learning exceeds a Cartesian view of skill optimization on several accounts: it is a dynamic, embodied, emergent and creative experience, in which processes and goals are intimately connected. Goals here are not determined destinations; they evolve and thus become emergent properties themselves. James P. Gee, a linguist focusing on learning, identifies 36 learning principles video gaming teaches, and indeed active learning tops his list, closely followed by situated meaning making, critical learning and collaboration.[66] These selected principles are indicative of the three streams in relation to which his work can be situated: situated cognition, new media literacy and implicit learning, as discussed above. The streams do not compete, but overlap and complement each other. In the context of this chapter, situated cognition illustrates the contextual and creative nature of (learning) experiences; implicit learning adds to the emergent non-conscious perception; cognition and decision-making complement their traditionally favoured conscious counterparts; and new media literacy focuses on the active, productive and social aspects of gaming.

Gee's active learning enriches the common notion of 'learning by doing' with the idea that people actively create meaning by selecting interactions that are intimately connected to their physical, social and temporal contexts. This situated perspective essentially defies the idea of learning by instruction and replaces it with an embodied account of learning. Damasio's somatic marker theory asserts that bodily physiological and cognitive processes in information processing are deeply intertwined, and emotions are crucial in driving efficient decision-making. John McCarthy and Peter Wright discuss embodied cognition in the context of HCI: in their account, the felt and embodied (or aesthetic)[67] experience is 'open and unfinalizable', 'a dynamic process of becoming open to the future',[68] i.e. one that is fundamentally contextual and creative. Pleasure here is affiliated with a fulfilling and meaningful aesthetic experience. Yet, although fulfilment signifies intense experiences that oscillate indivisibly between rules and uncertainty, pleasure and challenge, fun and frustration, the creative power, disappointingly, does not exceed the realm of experience. Here, the creative potential of emotions is reduced to shaping our perceptions, choosing how to react and, consequently, how to adapt. This take on evolutionary processes is reminiscent of the procrustean attempts of HCI in the 1980s to fit the user to the machine – then by instruction, now by reflexive adaptation.

Using Pine and Gilmore's model of 'experience realms',[69] one could say that McCarthy and Wright's aesthetic experience stops at the educational realm, where participants actively absorb experiences, but does not reach out into escapist experiences,[70] in which participants not only actively shape their experiences, but also their environment. The claim that the greatest potential for active and involved participation lies in the context of networked online communities and gaming[71] is echoed in Jane McGonigal's argument. For her, since the 'reality is broken' (where reality consists of uninspiring listless chores), there is a 'mass exodus' of those escaping into alternative (i.e. game) spaces, as they offer what we crave: engaged, fulfilling and blissful productivity.[72] Gee claims that active learning in gaming is always joined by critical learning, a concept that suggests that we not only understand systems, apply rules and creatively produce action within constraints, but we are also competent enough to critique and manipulate the meta-levels of the systems' design grammars.[73]

Design grammars consist of internal (e.g. rules, structures) and external (e.g. social practices, jargon) structures, both of which are composed of tangible and intangible components. Examples for such critical and social practices in gaming are plentiful and range from developing a variety of strategies to overcome challenges, such as glitches and bugs, to reaching out into the social space by reading or downloading cheats, patches and modifying (modding) games. Modding is an accepted and commercially encouraged form of collaborative hacking, in which peer produced hacks can be shared, downloaded and installed. Some companies offer built-in modding tools, while others provide source codes or code pieces (magic armour or sexy clothing for female characters in role games are popular choices).

Game production companies not only endorse players' individual creativity, but treat it as a source of innovation to be employed during production processes. In other words, they create a space for collaborative innovation which involves both players and designers. Beta testing is extensive and commonly open to the game community, not only to detect faults or test robustness, but to ignite and harvest players' critique and innovative ideas. Blizzard, the company behind *World of Warcraft*,[74] openly cultivates this practice as part of its innovation strategy, relying on a large-scale collaborative process as tens of thousands of their subscribers participate in beta gaming .[75] Blizzard executives are clear that this policy is not unproblematic and can produce frictions (third party comments sometimes challenge game developers). Yet, these tensions highlight the potential of users' creativity: an experimental study demonstrated that participating users not only generated more original ideas than expert desig ners, but also assessed innovative ideas differently from the company.[76] Apart from challenging the notion of creativity as a property belonging to expert desig ners, such participatory design spaces can act as examples of open and working collaborative experiences that engage players intensely through their creativity.

Given the above, Jenkins' stance becomes clearer: Jenkins claimed that focusing on the more obvious aspects of fun in gaming experiences ignored the wider consequences of a newly developing wider 'media literacy' – 'a cultural model of a digitally active, participative, collaborative and distributed working youth culture'.[77] Participants and consumers have developed not only into discerning recipients, but also competent co-creators. Even in 2005, 60 per cent of American teens could be considered media creators.[78] They not only produced audiovisual digital media, mash-ups or mods, but also used networked media to run campaigns, organize events or test innovative business models. Jenkins is very clear that his primary focus is not on the technical aspect of gaming, but the cultural implications this practice generates, and in particular new forms of collaboration and creativity: 'interactivity is a property of the technical; while participation is a property of culture'.[79]

Following from the above, it can be stated that games are not only appropriated to convey conventional teaching material, but deconstructed to arrive at new teaching models and principles that promote critical enquiry and participative and creative cultural practices. For HCI and UX, expanding the single dimension of fun into the multiple facets of engagement offers an opportunity to engage both with the challenges users face as they create and change digital culture, and the challenges and changes that these new media-literate user groups bring about.

Conclusion

The theme of science fiction used in this chapter serves both as a critique of the instrumental and rationalistic modelling of the human in HCI and a metaphor to illustrate a genuine belief that a playful exchange between science, technology and culture can become an exciting productive space, oriented towards innovation. Science fiction also deems the cybernetic dream of self-organization to be fictitious, as cybernetics regards happiness as an equilibrium. Instead, the metaphor I use aligns fun with the complexities of unpredictably and change. A richer understanding of the human can be achieved by drawing upon its evolutionary origins as they depict human species as complex and social organisms, driven to play, creativity and innovation. Looking at gaming shifts the focus in new media interaction from the mechanics of its production to the cultures it is producing. In turn, using fun could support the strands of HCI and UX that are oriented at first wave cybernetics in reconsidering their inherited reductive understanding of the scientific method and opening up to richer and more complex models.

Dealing with fun helps support the development of a richer model of engagement. Conceptually, it challenges rationalistic assumptions that creative thought is a sole property of logical and conscious processes. For HCI/UX practice, drawing upon fun's fickle potential of distraction leads to work on a multifaceted engagement model where options range from light-hearted and pleasant to intense, creative and active engagement. An expanded vocabulary should be used to support discussions of revised and enriched ideas of fun, pleasure, creativity, learning, surprise and exploration. Understanding emotions as active and driving forces and taking into account the evolutionary framework posing creativity as an evolutionary force encourages the adjustment of HCI and UX practices towards the collaborative design process of emerging participatory cultures. In addition, drawing upon game studies shows that the engaged experience does not only morph binary satisfaction into a qualitatively rich engagement, but also makes it become intertwined with learnability, a traditional key usability factor in interaction design.

Fun, in turn, emerges from the above as a multifaceted thing: connected to pleasure, emotion, distraction, absorption and obsession, unexpectedness and exhaustion, experiential learning and creativity, it resists dualisms and checklists and feeds upon the complexity of distributed experiences.

Notes

1 Bolter, David and Grusin, Richard*, Remediation, Understanding New Media* (Cambridge, MA; London: MIT Press, 1999), 9.

2 This can be claimed despite some acrimony between HCI's research and application oriented strands; see Meister, David, *The History of Human Factors and Ergonomics* (Mahwah, NJ and London: Lawrence Erlbaum Associates, 1999), 155.

3 Ibid., 148.

4 Dourish, Paul, *Where the Action Is* (Cambridge, MA: MIT Press, 2001), vii.

5 Hayles, Katherine, *How We Became Posthuman. Virtual Bodies in Cybernetics, Literature, and Informatics* (Chicago, IL and London: University of Chicago Press, 2000), 58.

6 See Card, Stuart K., Moran, Thomas P. and Newell, Allen, 'The Human-Processor Model', in *The Psychology of Human-Computer Interaction* (Hillsdale, NJ: Erlbaum, 1983), 24–6.

7 Preece, Jenny. *Human–Computer Interaction* (Wokingham: Addison-Wesley, 1994), 714.

8 'Terminator 1 develops relationship with Humans': http://blip.tv/confused-matthew/matthew-s-favorite-movies-terminator-2-judgment-day-part-2-6250047; 'Terminator's sacrifice': http://blip.tv/confused-matthew/matthew-s-favorite-movies-terminator-2-judgment-day-review-part-3-6253 (both accessed 26.11.2013).

9 Usability is a key concept in HCI and is defined as 'the extent to which a product

can be used by a specified user to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use' by the ISO 9241–11 standard.

10 This idea still conforms to a UCD methodology, as only the lowest level (L1) requires the consideration or training of the user; while the technology experts analyse, design and deliver systems. Eason, K., *Three Levels of User-centredness, Information Technology and Organisational Change* (London: Taylor & Francis, 1988) in Smith, Andy, *Human Computer Factors. A Study of Users and Information Systems* Information Systems Series (London: McGraw-Hill, 1997), 83; Nielson, Jacob, 'Noncommand User Interfaces', 1993, http://www.nngroup.com/articles/noncommand/ (accessed 26.11.2013).

11 Booth, Paul, *An Introduction to Human-Computer Interaction* (Hove: Erlbaum, 1995), 16, emphasis added.

12 See Dix, Alan John, *Human-Computer Interaction* (London: Prentice Hall Europe, 1998, second edition); Preece, *Human-Computer Interaction*; Sutcliffe, Alistair, *Human-Computer Interface Design* (Basingstoke: Macmillan, 1995, second edition).

13 Booth, *An Introduction to Human-Computer Interaction*, xii.

14 Clippit was an interactive animated character which – as part of Microsoft Office – interfaced to Office help content.

15 Clippit, http://en.wikipedia.org/wiki/Office_Assistant (accessed 26.11.2013).

16 Alan Cooper is perhaps best known as the 'Father of Visual Basic', or as the author of 'About Face (1/2/3)'. See Cooper, Alan, *About Face: The Essentials of User Interface Design* (Foster City, CA: IDG Books Worldwide, 1995); Cooper, Alan and Reimann, Robert, *About Face 2.0: The Essentials of Interaction Design* (New York: Wiley, 2003); Cooper, Alan, Reimann, Robert and Cronin, Dave, *About Face 3: The Essentials of Interaction Design* (Hoboken, NJ and Chichester: Wiley, 2007, rev. edn).

17 Cooper, Alan, *The Inmates Are Running the Asylum* (Indianapolis, IN: Sams, 1999), 126.

18 The technique was popularized for the interaction community in his book *The Inmates are Running the Asylum* (Indianapolis, IN: Sams; Hemel Hempstead: Prentice Hall, 1999).

19 http://www.adaptivepath.com/ideas/e000862 (accessed 26.11.2013).

20 Boehner, Kirsten, DePaula, Rogério, Dourish, Paul and Sengers, Phoebe, 'Affect: From Information to Interaction', in *Proceedings of the 4th Decennial Conference on Critical Computing: Between Sense and Sensibility* (Aarhus, Denmark: ACM, 2005).

21 Picard, Rosalind W., *Affective Computing* (Cambridge, MA and London: MIT Press, 1997).

22 Norman, Donald A, *Emotional Design: Why We Love (or Hate) Everyday Things* (New York: Basic Books, 2004).

23 Garrett, Jesse James, 'The Elements of User Experience' (2000), http://www.jjg.net/elements (accessed 12.06.2008).

24 In 2009 the ISO 9241–11, the standard for HCI usability criteria, was revised and expanded to incorporate the User Experience (ISO FDIS 9241–210:2009); Ergonomics of human system interaction-part 210: Human-centred design for interactive systems (formerly known as 13407). International Organization for Standardization (ISO), Switzerland.

25 Bevan, Nigel, *What is the Difference between the Purpose of Usability and User Experience Evaluation Methods?*, UXEM '09 Workshop, INTERACT 2009, Uppsala, Sweden, 2009.

26 Hassenzahl, M. and Blythe, M., 'The Semantics of Fun', in *Funology: From Usability to Enjoyment*, Mark A. Blythe et al. (eds) (Dordrecht and London: Kluwer Academic, 2004), 91–100.

27 Follett, Jonathan, 'Engaging User Creativity: The Playful Experience', (17 December, 2007), http://www.uxmatters.com/MT/archives/000252.php (accessed 12.09.2008).

28 Huotari, Kai and Hamari, Juho, 'Defining Gamification – a Service Marketing Perspective', in *Proceedings of the 16th International Academic MindTrek Conference 2012* (Tampere, Finland, 2012).

29 Pavlus, '62 Reasons Why 'Gamification' is Played Out', Fast Company, 2010, http://www.fastcodesign.com/1662656/sixty-two-reasons-why-gamification-is-played-out (accessed 26.11.2013).

30 Radoff, Jon, 'Gamification', http://radoff.com/blog/2011/02/16/gamification/ (accessed 26.11.2013).

31 Ferrara, John, 'The elements of player experience', http://uxmag.com/design/the-elements-of-player-experience (accessed 29.03.2014).

32 Ferrara, John, *Playful Design – Creating Game Experiences in Everyday Interfaces* (Brooklyn, NY: Rosenfeld Media, 2012, 1st edn), 33.

33 Early writing on HCI recommended focusing user's attention on particular aspects of the interface by making them blink, flash or animate; Maguire, 'A Review of Human Factors Guidelines and Techniques for the Design of Graphical Human–Computer Interfaces', in *International Journal of Man–Machine Studies*, 16(3) (1982): 237–61, cited in Preece, *Human-Computer-Interaction*, 58.

34 Nielsen, Jakob, *Designing Web Usability: The Practice of Simplicity* (Indianapolis, IN: New Riders, 2000), 9.

35 Harrison, S., Tatar, D. and Sengers, P., 'The Three Paradigms of HCI', *Proceedings of CHI 2007*, alt.chi, San Jose, CA, May 2007.

36 Csikszentmihalyi, Mihaly, *Flow: The Psychology of Optimal Experience* (New York: Harper & Row, 1990).

37 Hassenzahl and Blythe, 'The Semantics of Fun', 98.

38 Ibid.

39 Ibid.

40 Lazzaro, Nicole, 'Why We Play Games: Four Keys to More Emotion in Player Experiences', presented at Game Developers Conference (San Jose, 2004).

41 Chatfield, Thomas Edward Francis, *Fun Inc.: Why Games Are the Twenty-First Century's Most Serious Business* (London: Virgin, 2010), 50.

42 Hassenzahl and Blythe, 'The Semantics of Fun', 98 .

43 Pine, B., Gilmore, Joseph and James H., *The Experience Economy: Work is Theatre & Every Business a Stage* (Boston: Harvard Business School Press, 1999), 97.

44 Hassenzahl and Blythe, 'The Semantics of Fun', 10.

45 For the purposes of this article *flow* will refer to Csikszentmihalyi's concept as identified by the use of italics.

46 'It is the full involvement of flow, rather than happiness … When we are in flow, we are not happy, because to experience happiness we must focus on inner states, and that would take away attention form the task at hand. […] We can be happy experiencing the passive pleasure of a rested body, warm sunshine, or the contentment of a serene relationship …' Csikszentmihalyi, *Flow,* 32.

47 Csikszentmihalyi's experience sampling method (ESM) has been repeatedly criticized because it relies solely on (admittedly vast amounts of) self-reported accounts of participants, i.e. leaves plenty of room for ambiguity and interpretation. The ESM involves participants carrying a pager or beeper watch and responding to items on a self-report form when 'beeped' (e.g. what they are doing, level of challenge, level of skill, affect). See Ellis, G., Voelkl, J. and Morris, C., 'Measurement and Analysis Issues with Explanation of Variance in Daily Experience Using the Flow Model', in *Journal of Leisure Research,* 26 (1994): 337–56; Marr, Arthur J., 'In the Zone: A Biobehavioral Theory of the Flow Experience', *Athletic Insight, the Online Journal of Sport Psychology*, 3(1), (2001).

48 Ashby, F., Isen, A. and Turken, A., 'A Neuropsychological Theory of Positive Affect and Its Influence on Cognition', *Psychological Review*, 106(3) (July 1999): 529–50.

49 Csikszentmihalyi, *Flow,* 72.

50 Koepp, M. J., Gunn, R. N., Lawrence, A. D., Cunningham, V. J., Dagher, A., Jones, T., Brooks, D. J., Bench, C. J. and Grasby, P. M., 'Evidence for Striatal Dopamine Release during a Video Game', *Nature*, 393(6682) (May 1998): 266–8.

51 In order to facilitate an MRI scan during game activity the game had to be simple.

It is agreed that the findings might not scale into complex video games, and that more studies are necessary to understand ethnic, cultural and genre differences.

52 Hoeft, F., Watson, C. L., Kesler, S. R., Bettinger, K. E. and Reiss, A. L., 'Gender Differences in the Mesocorticolimbic System During Computer Game-Play', *Journal of Psychiatric Research*, 42(4) (Mar 2008): 253–8.

53 McGonigal, Jane, *Reality Is Broken: Why Games Make Us Better and How They Can Change the World* (London: Vintage, 2012), 43.

54 Chatfield, *Fun Inc*, 10.

55 Ibid., 11.

56 Ibid., 82.

57 Usability is the ease of use, learnability and satisfaction in interaction; a key concern in traditional HCI, see Preece, *Human Computer Interaction,* 1994.

58 Chatfield, *Fun Inc.*, 37.

59 Ibid., 9.

60 Ibid., 4.

61 Claxton, G., 'Investigating Human Intuition: Knowing without Knowing Why', *Psychologist*, 11 (1998): 18–19.

62 Damasio, A. R., *Descartes' Error: Emotion, Reason and the Human Brain* (London: Picador, 2005). Damasio's somatic marker hypothesis acted as source material in the context of HCI for Don Norman's *Emotional Design*and Roz Picard's *Affective Computing*.

63 Damasio, Antonio R., *Looking for Spinoza: Joy, Sorrow and the Feeling Brain* (London: Heinemann, 2003), 150.

64 Claxton, 'Investigating Human Intuition'.

65 In case HCI or UX designers wonder how this affects digital interaction design, they might refer to Susan Weinschank's 'Neuro Web Design', which speaks of an 'intelligent unconscious' that is 'faster and smarter than your conscious mind' in the context of internet interaction. See Weinschenk, Susan, *Neuro Web Design: What Makes Them Click?* (Indianapolis, IN: New Riders, 2009), 10.

66 Gee, James Paul, *What Video Games Have to Teach Us about Learning and Literacy* (New York: Palgrave Macmillan, 2003).

67 As in Dewey's notion of the aesthetic and prosaic experience. It is important to understand that the aesthetic experience in this context does not refer to art, art objects or 'a museum conception of art', but the potential of everyday (prosaic) experiences being transformed into enriched, meaningful, and whole; in McCarthy and Wright, *Technology as Experience*, 58.

68 Ibid., 69.

69 Pine, Joseph and Gilmore, James, *The Experience Economy,* Figure 2.1.

70 Ibid.

71 McGonigal, Jane, *Reality Is Broken*, 34–5.

72 Ibid., 51.

73 Gee, *What Video Games Have to Teach Us about Learning and Literacy*, 32.

74 *World of Warcraft* (*WoW*) is currently the world's most-subscribed massive multiplayer online role-playing game with 11.4 million subscribers as of March 2011; it holds the Guinness World Record for the most popular MMORPG. In April 2008, *WoW* was estimated to hold 62 per cent of the MMORPG subscription market, see http://en.wikipedia.org/wiki/World_of_Warcraft (accessed 26.11.2013).

75 See http://www.curse.com/articles/blizzard-entertainment-en-news/25641.aspx (accessed 26.11.2013).

76 Kristensson, P., Magnusson, P. R. and Matthing, J., 'Users as a Hidden Resource for Creativity: Findings from an Experimental Study on User Involvement', in *Creativity and Innovation Management*, 11 (2002): 55–61. doi: 10.1111/1467-8691.00236.

77 Jenkins, Henry, *Fans, Bloggers, and Gamers: Exploring Participatory Culture* (New York and London: New York University Press, 2006).

78 They created blogs or web pages, posted original art work, photography, stories or videos, or remixed online content into their own new creations. Most have done two or more activities, see Lenhardt, A. and Madden, M., 'Teen Content Creators and Consumers. Washington, DC: Pew Internet & American Life Project' (2 November 2005), http://www.pewInternet.org/PPF/r/166/report_display.asp in Jenkins, Henry, *Convergence Culture: Where Old and New Media Collide* (New York and London: New York University Press, 2008).

79 Jenkins, *Fans, Bloggers, and Gamers*, 8.

11

A Fun Aesthetic and Art

Annet Dekker

Introduction

The neon sculpture *HAHA HIHI* (1993) adds a refreshing note to the cacophony of advertising signs and tacky shops in Schiphol airport. *HAHA HIHI* is a chandelier by John Körmeling and instead of using candles it shows the words 'HA' and 'HI' in bright neon colours. Körmeling has been called an architect, a visual artist, an inventor and a freethinker. The range of his work is equally diverse: light sculptures, comic strip-like drawings with visual and verbal witticisms, sketches of viable and imaginary structures, models and unconventional solutions to spatial planning problems (such as *Happy Street* for the World Expo 2010 in Shanghai). Körmeling is at his best in the public space and urban planning areas. His unique working style and surprising perspective on everyday things convey the relativistic humour that emanates from his works. Körmeling exposes the absurdity and banality of everyday life by means of rapid associations. He makes complex citations understandable and simple facts grotesque. *HAHA HIHI* laughs at the people who move like ants through the airport. At the same time it encourages people to pause and reflect on their environment and their behaviour in the building. Evoking a smile or a frown even if just for a moment, Körmeling's works are simple yet rigorous, and his tactics are always friendly.

In another location, on deserted landfills in Berlin, near rubbish bins, garbage bags and refuse, computer screens suddenly flicker into life. The old, grey and now clunky looking cathode ray tube (CRT) screens show net art works: *C.R.E.A.M.* (2010) by Evan Roth, *therevolvinginternet.com* (2010) by Constant Dullaart, *Super Mario Clouds* (2002) by Cory Arcangel and, the oldest of the quartet, *404* (1997) by JODI. This is Aram Bartholl's *Highscreen* (2011), a public intervention, in which he revived computer screens that had been dumped in Berlin's wastelands. Bartholl is well known for his projects and performances, which try to interestingly displace phenomena of the digital world into the physical environments alien to them. Although it is generally acknowledged that digital structures undergird and define the running of society in ways that make the distinctions between the digital and the offline obsolete, the shift between the digital and the physical material unfolding in its functions in relation to many other histories, practices, tendencies and habits may be seen as a conflictual, revealing rupture. From this perspective, Bartholl's main interest is in how digital communication and algorithms construct and change the physical space and vice versa, especially if one alienates the digital by imbuing it into a material other than the one of zeroes and ones. Works such as *Speech Bubble* (2007), in which volunteers with a large, speech text balloon on a stick trail people, or *Tweet Bubble Series* (2009), featuring Twitter texts that can be attached to a shirt, emphasize the limitations of the overt publicness that is prevalent on the internet, but looks absurd if the utterances are presented as physical objects wrapped around human bodies. With these works Bartholl shows how the digital and physical worlds can be seen as disconnected and how translations from one to the other lead to estrangement – unless you are in the game.

Unlike Bartholl's previous works, *Highscreen* (2011) is less playful and acts as a cautious investigation of the ways in which computation becomes core to our lives.[1] Although it still thrives on incongruence, *Highscreen* is also more subtle. The computer screens are old and have been discarded, but it is not just the plastic and metal that is thrown out, the art that goes with it is also left in the gutter. Some might consider it ironic that artworks that were never properly valorized by any official art institution end up as waste, materializing the metaphor, to be further destroyed forever. The work could be seen as just that: an act of irony, or perhaps dark humour. But the blinking 404 error message on one of the CRT screens will look familiar to many as the standard response code indicating that a website could not be found, and some people will be able to identify it as a work by JODI, the artist duo known for the subversive tweaking of software and exploration of the computer's back-end. As such I would like to argue that for those who recognize the works on the screens, *Highscreen* moves from being ironic to becoming fun. What is fun then and in which ways is it more complex and multilayered than humour and irony? A more interesting question, perhaps, would be to see if fun is part of an aesthetic method, or has aesthetics of its own? Are these examples merely clever uses of a critical strategy, or do they reflect a broader cultural sensibility that is lost on those who are not 'in the game'? Does such sensibility only become manifest in the complete and finalized work as an 'effect', or could fun also be seen as characteristic of the artistic practice, the process itself?

An aesthetics of fun

When writing about fun, one has to treat humour cautiously. A subject of many investigations, the question of humour goes back a long way.[2] The notion of fun that I would like to explore does not exclude humour, the joke or playfulness; however, it does not stop there. What I am looking for is the kind of fun that reflects on and analyses processes to provide insights and inspire new thoughts. Fun in this sense relates to the fun of making and sharing. It is the fun that takes place processually and practically, as well as conceptually to arrive at startling events, projects and practices. The aesthetics of fun resides in such processes, whether prepared or accidental, spatial or code-based, terminal or open-ended, across scales, strata and time.

The installation *HIHI HAHA* first prompts an exploration of the aesthetic of fun by highlighting Körmeling's interest in presenting work in places where people are not prepared for controversies, or where they can be taken by surprise. Although he is commenting on society, he is not criticizing the qualities of the material he uses or the relationships such material creates, nor is he opening up the production process for others to use or learn from. What matters for Körmeling is how the final presentation of the work functions in its context, which can be and often is funny. However, this is only one aspect of the aesthetics of fun. It is the repetitiveness and perseverance of Körmeling's approach, which can be seen in other works such as *Happy Street* or *Rotating House* (2008) (a freestanding Formica house rotating on a roundabout), that the strategy fully reveals itself. He seems primarily interested in staging confrontations between the expected and the grotesque through shock. Employing this as a method of presentation and performance, as well as perception, *Highscreen* seems to come slightly closer to what can be discussed as aesthetics of fun. To unpack this further I will analyse the works that are presented as part of *Highscreen* to see how they convey an aesthetic of fun.

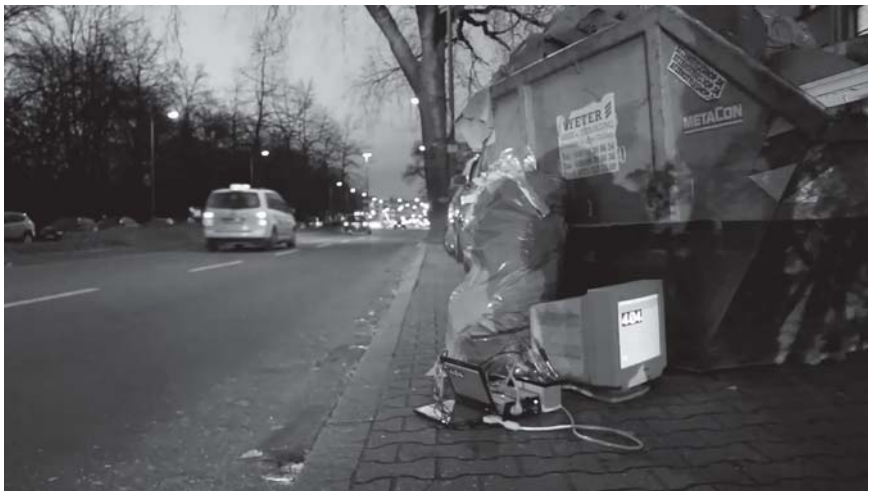


Figure 11.1 Aram Bartholl, *Highscreen* [JODI] (2011)
Source Photo by Aram Bartholl

404

JODI, the Dutch/Belgium duo, achieved fame in the mid-1990s and continued to produce work that is hated, loved and misunderstood. *404* is a very good example of such work. The website opens to display a page titled *%20Wrong* and the '404 message' in the webpage, creating a short-lived panic in casual netsurfers, who can turn away, but those more familiar with net art projects can look for entrances. Some random clicking reveals that each of the numbers in '404' leads to new screens: unread, reply and unsent. The sequence isn't very promising. Nevertheless, using the reply option, *Re:* in the *%20Unread* page, will lead to a response being delivered – an encrypted version of the email that the visitor has just sent. What appears is a text without vowels, followed by a string of consonants and numbers that sometimes resemble words (the *sht* and *fck* are easily grasped). *%20Unread* relates to two things. On the one hand, it shows the act of censorship that is common on chat boards by using the same technology CGI (Common Gateway Interface). The CGI protocol returns the results to the web server in the form of a standard output, in this case without vowels and with additional underlying text. On the other hand, the output of the CGI can be seen as an act of Dadaist or abstract poetry.

Much as Dadaist, Constructivist and Futurist poetry was about liberating fonts and language from their respective constraints, so Piet Mondrian advocated a pure 'plasticism', stating that '*Music* tends toward the liberation of *sound*, *literature* toward the liberation of *word*'.[3] Now programming is heading towards the liberation of computation. JODI release software and create openings through which it can pass on its 'way to freedom'.[4] Rhythm, syntax and letterforms were central to 'plasticism', which used punctuation marks and unusual text layouts to indicate how the text should be read or, better, voiced. Typography became an active visual instrument that emphasized form, meaning or sound. Similar strategies can be found in the work by JODI, where punctuation marks lose their neutrality and start speaking themselves. JODI's abstract visualizations of code and use of punctuation comes very close to the *Typo-plastieken* (1925) of Pietro (de) Saga. Pietro (de) Saga, pseudonym of Stefi Kiesler, made several Typo-plastieken and Dactyloplastiek (1925) by repeatedly typing letters over each other to create visual forms. She sublimated the meaning of a letter to its abstract, visual image.[5] Her works, as much as JODI's works, can be appreciated best in conjunction with the Concrete poetry of the 1950s. Concrete poetry focused on language primarily as a material; concerned 'less [with] an understanding of meaning than an understanding of arrangements'.[6] JODI are also focused on computational arrangements, the structures of communication that perform at different levels and on extracting the meaning from non-significatory material, the processual performances of code.

JODI have a keen interest in protocols and re-use them in open ways. The protocol of *404* is the CGI, which is the focal point of the work and its structuring framework. The second page of the work *%20Reply* displays the internet protocol address (IP address) of all the entries that have been made against a black backdrop. Certainly, nothing is just what it seems either: the text typed by the user appears beside its IP address, also in black. Here people can read all the entries that were made, though only if they are quick as the screen reloads after a few seconds to display the first page. Anonymity is cast aside by showing the IP address in bright green.

The final page *%20Unsent* is a repetition of the text that was typed in 'reply', but this time the vowels become visible: these are the remnants of what was not sent before. Below the text is an abstract image, concrete poetry, where little running rabbits (which link to the main page), numerals and punctuation marks create a visual that is recognizable, although meaningless. The meaning of *404 not found* can only be understood by taking into account its network and software contexts. *404* works the protocol into the absurd. In the end, the

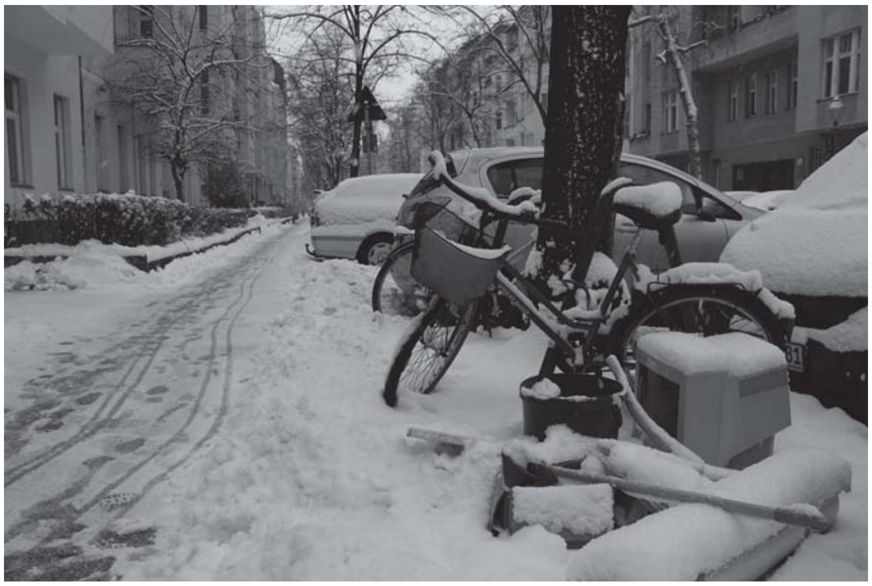


Figure 11.2 Aram Bartholl, *Highscreen* [Cory Arcangel] (2011)
Source Photo by Aram Bartholl

magician's rabbit runs back, leaving the user confused, enacting and re-enacting the ridiculousness and exposure of browsing, emailing, digital work and leisure.

JODI's work reflects on the structures and agreements that construct computational networks; and it is through the absurd that the political and aesthetic emerges out of code, software gestures, language and digital habits. JODI poke around, deleting brackets from markup language, unravelling the material. The aesthetic of fun here is twofold, appearing in JODI's intentional methodological (ab)use of the material, which in its working fortifies the frustration of using and cruel pleasure of dispatching a perfect machine.

Super Mario Clouds

Another computer screen displays bitmapped white clouds that are endlessly inching across a blue sky. The work is *Super Mario Clouds* (2002) and it is one of Cory Arcangel's early works. From the start of his career he has shown an interest in low-key techniques, outmoded computer games and early electronics. *Super Mario Clouds* is based on the legendary *Super Mario Bros* game released in 1985 by Nintendo. Arcangel hacked the game and modified it so that the only visible elements are the blue sky and the white clouds moving across the screen. The main protagonist, Mario, all the obstacles and the landscape have been removed. Does this make the work fun? It might evoke a smile or a frown in people recognizing the original game, who may wonder if it is an act of rebellion against the violence, or competitiveness of gaming, or 'just a joke'. We can draw some other characteristics of the fun aesthetic out of it.

In 2002 Arcangel wrote the program, burned it on a chip, hacked an old Nintendo game cartridge, de-soldered one of the Nintendo graphics chips in the cartridge and soldered his new chip in its place. Because at the time it was not yet possible to put the video online, he made a gif that was distributed with a tutorial (complemented with humorous and informative comments) that explained his motives for making the work and included the source code behind the work as well as the instructions on how to construct the object.

In 2004, two years after the release, Arcangel made a short instructional video that explained, step by step, the creation of the work. The *Making of Super Mario Clouds* is made as an amateur video without slick graphics, camerawork or soundtrack. It merely shows Arcangel in his apartment making a cartridge; his intention, as he explains, was to give viewers 'a feel for the process, in all its gloriously boring true detail'.[7] The project was in line with projects that various members of the Beige [Programming Ensemble] were doing. Beige described themselves as an electronic music recording company and computer programming ensemble. Arcangel joined the Beige collective in the late 1990s.[8] Their concern was to create a wider acceptance of media art and democratize computer hardware and software. Moreover, they critiqued the stylized, restricted and pre-determined parameters of the machines that were shaping the 'hyper-technologized' world. Their philosophy was 'Intentional Computing', 'work in which the artist demonstrates a complete understanding of the machine he/she is composing on [from the CHIP to the display]'.[9] Through hacking and modifying existing hardware and software they tried to stretch the allowances of the possible. In other words, they wanted to re-tune the unison in which creating artwork performs together with the tool or medium being used. It was about tunnelling into the computer and starting from the inside.

This approach does not significantly differ from that of JODI: both try to destabilize standardized and habitual environments produced by dominant, commercial computational systems. However, Arcangel's main focus is on form and colour (and, in his more recent work, rhythm). Whereas JODI reveal code as part of the work, Arcangel supplements the projects with tutorials and video documentaries detailing the creation and functioning of the work. However, the work itself, in its minimalist style, functions better than JODI's as a traditional work of art. Arcangel brings something 'human' to the computer by ensuring that the work is never too slick and always has an ironic twist that many people will recognize. This could be seen as a populist approach. Whereas JODI take the 'humanity' out of the computer, making it 'fun' for those immersed in the processes constructing the computational functionality, Arcangel facilitates

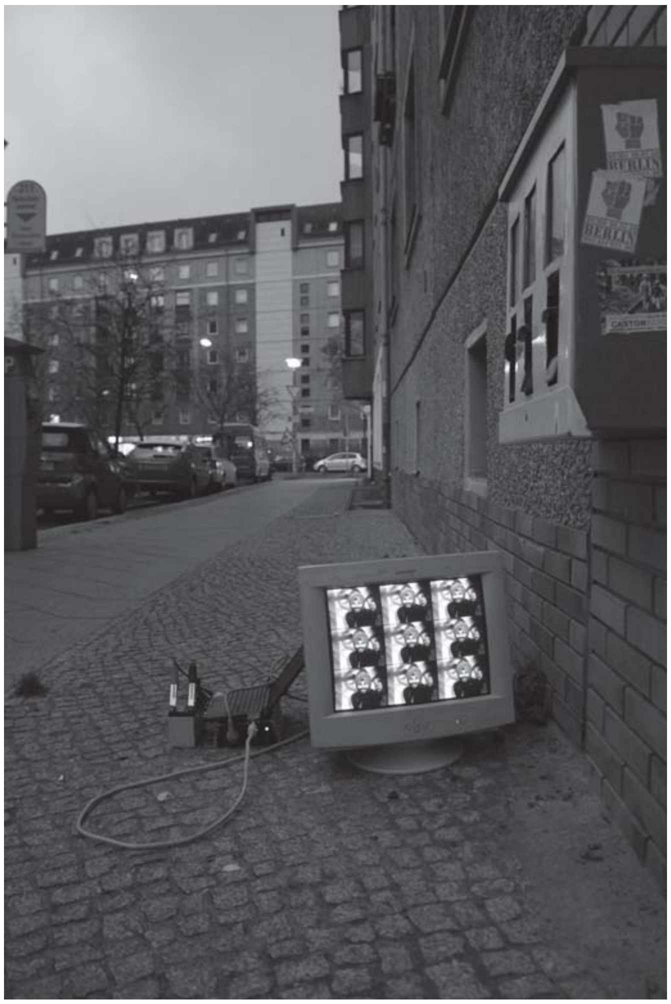


Figure 11.3 Aram Bartholl, *Highscreen* [Evan Roth] (2011)
Source Photo by Aram Bartholl

the work for the viewer. JODI double and triple frustration, whereas Arcangel works in the line of pop art and minimalism. When the art world met JODI, it did not know how to react, but when it saw Arcangel's work, it recognized the 'fun' and abstraction reminiscent of Andy Warhol.

But there is a more significant difference between the two artists, which concerns the experiencing of the work and the participation of their audience. JODI's work can be confusing, but although Arcangel presents his work as open and free to use, presumably not many people followed his instructions. Arcangel does not sound optimistic himself recalling that the only reaction Beige received upon the release of their music album *The 8-Bit Construction Set* (2000), which contains programs, was from 'a couple of kids from the Netherlands, and they were like "cool!" – but that's about it. (...) There are only about five people in the world who have really ever done that [loading the cassette on to a computer]'.[10] Do people take part by witnessing, watching, looking or relaying?[11] It can also be argued that JODI do not offer their audience the pleasure of looking at a nice image, but it is through actively engaging with their artwork, and doing the work of linking it with the computational environment in which it thrives and on which it comments, that an appreciation can emerge – an instructional exercise in its own right.

It is the exploration and the experience of the manual gesture – the doing it yourself – that resonates here. These processes are 'forced' by JODI, while Arcangel leaves it to the whims of the visitor. Both, though differently, employ the fun of working, touching, tinkering with something: the Do It Yourself (DIY) fun. The DIY attitude, facilitated by the Web, has given rise to a number of art initiatives that manifest themselves through online platforms. One person behind such an initiative is Evan Roth, co-founder of F.A.T. Lab, the Free Art and Technology Lab, and the author of the third work of *Highscreen*, *C.R.E.A.M. (Cache Rules Everything Around Me)*.

C.R.E.A.M. (Cache Rules Everything Around Me)

C.R.E.A.M. by Evan Roth is a hypnotic collage of hundreds of graphics interchange formats (GIFs) found on the internet. Mashed together in a nine-minute video clip, they portray the internet at a particular moment in time. Single, animated GIFs often function as memes, cultural artifacts that spread virally on the networks. The term 'meme' was suggested by Richard Dawkins in the

The Selfish Gene (1976)[11] to signify a unit of cultural information spreading and surviving analogously to a gene.[12] In the tradition of memetics, Blackmore argues that beliefs and ideas are replicators, memes, which are copied from person to person and define people's behaviour.[13] She notes that technically everything on the internet can be seen as a meme, as the information circulating around is variable, selected and copied.[14] The notion of the meme as a replicator is core to Roth's practice and can be usefully explored in relation to the aesthetics of fun.

Roth started working as an artist around the same time as Arcangel and has also been involved in working in collaborative environments. He co-founded F.A.T. Lab, with James Powderly in 2007.[15] It is impossible to offer a single definition of the lab; generally it is thought to be an ongoing experiment, conducted by artists, engineers, scientists, lawyers and musicians alike, that facilitates innovation, collaboration and critique. In the early years F.A.T. Lab was mostly visible in the United States and formed around a group of research fellows, artists in residence or others affiliated with Eyebeam Art and Technology Center in New York. The name F.A.T. Lab is reminiscent of E.A.T., the non-profit organization Experiments in Art and Technology[16] that was launched in 1966 during the organization of *9 Evenings: Theatre and Engineering*, a series of performances uniting artists and engineers.[17] In an attempt to make people more active in the becoming of the technologically defined society, E.A.T. tried to bridge the gap between disciplines and specializations. Although this agenda is important to F.A.T. Lab, their approach centres on the specific usages of technology in the public domain by engaging with software and the questions of openness.

Whereas E.A.T. focused on the creation of large technological performances and installations, showcased in the design for the Pepsi Pavilion at the Expo '70, and resulting in a remarkable spectacle of artistic and engineering wizardry, F.A.T. Lab explores their material and presentational platforms to produce Firefox add-ons, plug-ins, silly objects or political GIFs. What can be read as a footnote to changes brought about by time, the emergence of software and the further re-configuration of the disciplinary formations, copyright, control and ownership arrangements becomes core to F.A.T. method. F.A.T. Lab examines the Web and takes advantage of the available material, writing small scripts and programs to retrieve and remix information. F.A.T. tweaks existing data and tools to comment on the way the Web is governed and can be used. As such it is the fun of 'slight touch' and sharing that becomes another aspect of the aesthetic of fun.

The slight touch does not indicate that the political ideas of F.A.T. Lab are naive. F.A.T. has a strong position on the use and support of free and open source software. Code written by F.A.T. is open for others to use, is documented and improved upon during lectures and workshops. In fact, all the group's processes and artworks become open and as such rely on other people's input, as in open-source project development.[18] Theo Watson, one of the members, says:

> [it] is interesting to see this process being applied to Internet art, experimental projects and web memes. It is like open source project development but for those people more interested in pop culture and art than in programming software tools. It is open source for the masses.[19]

The 'lightness' of their approach also becomes manifest here, where the political starts acting virally. It is propagated and practised through memes (qualitative meme modulation)[20] and as such can be seen as a tactical way of moving through mass and corporate culture.

It is through the active use of the meme as a method that F.A.T. Lab adds another dimension to the aesthetics of fun. Similarly to the above-mentioned group Beige, F.A.T. Lab uses existing amateur material that is remixed, adapted or merely re-contextualized. The result is always light hearted and, above all, fun. This is the fun of making, the fun of idiocy[21] and nonsense that acts by topping the noise. The nature of memes as replicators is taken as a starting point and radically exploited. Using social networking platforms, F.A.T. Lab infiltrates the Web by being 'dedicated to enriching the public domain one mutha-fuckin LOL at a time'.[22] They achieve this by creating simple 2D glasses to escape the 3D hype or setting up a *Speed Project* site with a special widget to track time. The website demonstrates how to do as much as possible in under eight hours (a normal office working day), actively engaging in the mechanics of acceleration. Furthermore, members of F.A.T. try to mould the behaviour on online platforms by inviting others to comment and make their own additions, which can all become an art project in its own right. One such example is *White Glove Tracking* by Roth and Ben Engebreth that started in 2007. For this project they asked internet users to help isolate Michael Jackson's white glove in the performance of Billy Jean. Rather than writing unnecessarily complex code to find the glove in every frame of the video (10,060 frames in total), they asked the audience to simply click and drag a box around the glove in one frame and upload it to their site. The ridiculous task of isolating the white glove allowed the data collection process to become fun and viral. The data set was released

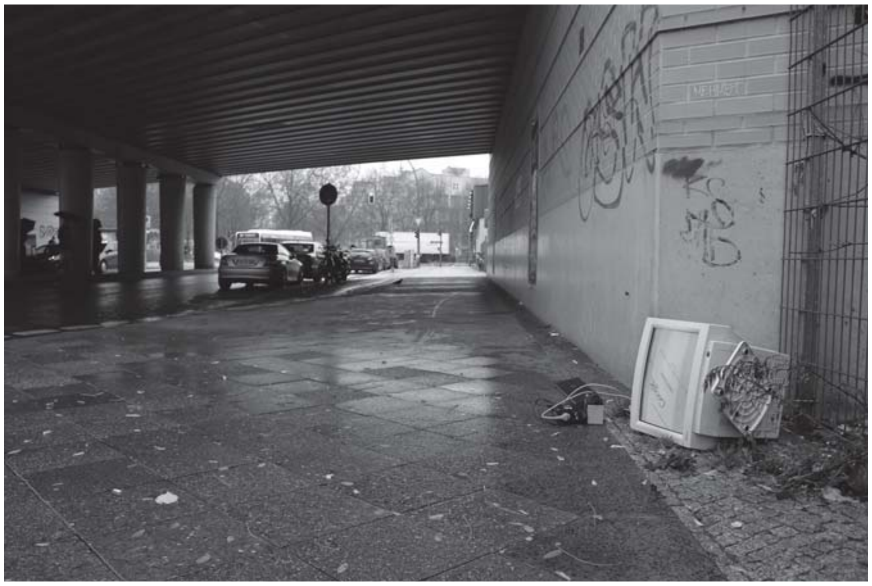


Figure 11.4 Aram Bartholl, *Highscreen* [Constant Dullaart] (2011)
Source Photo by Aram Bartholl

as a text file as well as a Processing and Open Frameworks project for others to use. Although it can be argued that the use of memes and silliness is a strategy, a formula or a method, as with Körmeling, functioning by conspicuously tracking down what is available on the Web both technically and culturally and reclaiming it for their purposes, here idiocy and nonsense forcefully enter the aesthetics of fun.

therevolvinginternet.com

The Revolving Internet (2010) by Constant Dullaart engages the similar fun of idiocy and nonsense. The work is a clever trick, which places Google's homepage in an iframe with Javascript changing the document object model (DOM) values, which causes the page to rotate on the screen. The work functions well in *Highscreen* where among the rubbish on the side of the road one can see the Google search engine interface tumbling around as if twisted by a whirlwind from a passing truck or bike. It is also a good example of Dullaart's practice. His method lies in identifying recurring themes and motives of the computational practice and culture and re-contextualizing, alienating them.

Often, Dullaart transforms banal images or tiresome website interfaces just enough to highlight some of their inherent 'strangeness'. For instance, he has a collection of links to domain names on sale that no one would buy. Linked together by their shared status of semi-existence, they form the rubbish dump of the internet. In his Delicious account, Dullaart labels the poetic domain names, like www.baddomainname.com and www.hopeless.org as 'ready mades'. His fascination goes further to dwell on the design of the 'default settings' of these sites. His 'ready mades' are, therefore, a reflection on the omnipresent stock photography and insipid typefaces.

With his practice Dullaart follows in the footsteps of previous net artists who elevated the formal aesthetic or unfrequented corners of the Web into art.[23] By deconstructing the mundane in a playful, visual way Dullaart engages with contemporary internet culture indiscriminately. While being playful, his works also celebrate a certain coarseness that is practised on the networks and inverted to be thrown back at the audience.[24] Although employing a methodology of fun that draws upon the absurd, the banal and the nonsensical, similarly to the F.A.T. Lab, it is primarily the way in which Dullaart positions himself in the landscape of the Web where the aesthetics of fun is most pronounced: as a member of surf-clubs. Surf clubs are the early twenty-first century's answer to the rise of the Web 2.0. Whereas the earlier internet generation communicated mainly through listserv, using mailing lists, the 'digital natives' set up their own clubs.[25] This new generation of artists uses devices such as continuous postings, real-time involvement and commenting on existing social network platforms, such as YouTube, Flickr or Facebook, while trying to define and maintain a shared aesthetic and a group identity. Surf clubs often use the blog format. Nasty Nets, Supercentral and Loshadka are small 'gathering' sites to which members upload found material, while others can comment, either textually or with images or video. Special software was written to simplify the upload of material, but the generic comment also functions as an essential component, becoming an aesthetic exploratory tool. The chronologically arranged commentaries form larger compositions or 'lists' of images.[26] The artworks that arise from these collective processes are situated in continuum with other works, references and commentaries.

Constant Dullaart, as contributor to several surf clubs, emphasizes the shared and collective experience of this iterative process that exhibits an aesthetic of fun. It becomes evident in his YouTube series (*You Tube as a Subject,* 2008), in which he works with the icons that appear embedded in YouTube videos.

Positioned against the black background of a loading YouTube video, the image of a play button trembles as if in a sudden earthquake, bouncing from side to side, changing colours and strobbing like a mini disco lightshow, falling down off the screen, and finally fading into a blur. Keeping in line with the spirit of the comment and the pressure to be quick, soon after Dullaart uploaded his series, Ben Coonley responded with a new series, this time taking the dots that signal the loading time of a video as his subject. In turn, for an exhibition in 2009, Dullaart versioned this visual discussion by creating a physical copy of the loading dots, *You Tube as a Sculpture*.[27] This time eight Styrofoam balls were hung in a circle against a black background, lit one after the other by eight spotlights and a simple disco light mixer. While the visitors were given the feeling of entering a loading YouTube video, they also started filming the balls and uploaded them to YouTube, thus, according to Dullaart, 'completing the circle of production and reproduction'.[28] The engagement first occurs online, then offline and returns again to the servers: 'The success of the sculpture meant that audience members documented the sculpture and finally became the uploading medium for my participation in the visual discussion set in motion by *You Tube as a Subject* a few years earlier'.[29] This circulatory human–machine process is fun found in the making across scales and by varying actors, who try to outwit each other, become the most original, but also by responding to each other and doing things together.

> The comments were one of the biggest compliments I have received in my life. I felt like I really needed to make the videos and upload them before someone else did. But when the comments came into my inbox late at night, all seven of them one after another, notifying me of the responses made by Ben Coonley, I knew I had hit a nerve. I laughed myself to sleep. It was a great surprise.[30]

The value and meaning of surf art relies solely on the distributive qualities – the network of other people's content and the means of its production and distribution.[31] It is in engaging with the material and the networks and employing and exploiting the structures such as memes that the art of surf club activity is constituted.

The most encouraging aspect of surf art is its distributive quality, the way in which it offers a new working practice, signalling a new aesthetics – one that is grounded in fun. Such fun has consequences for notions of authenticity and authorship.[32] Dullaart states:

> The authenticity and authorship of the original seems to be less clear when a response or another version of the original start to co-exist, but the general

> impact of the shared idea becomes larger. The online tradition of linking back to the origin of the idea (the original post or video) is an important one, of course, but sometimes the response is more valuable than the original.[33]

These artists deal with iteration, versioning and repetition as part of a distributed practice. The quest for originality is still important, but is achieved in a different way, for example, by being the first to comment with a brilliant and funny idea or being reblogged. Such authorial recognition is often monitored by the larger group.[34] Commenting also becomes a mechanism for establishing individuality, as participants combine shared meanings and tweak the shared parameters of the group in idiosyncratic ways. The structure of gaining and evaluating acknowledgement and esteem becomes rested within the devices of social media.[35] Here, the aesthetics of fun turns into an aesthetic of (gaining and loosing) control.

A fun aesthetics

All the projects of *Highscreen* work with the aesthetics of fun. It ranges from, but is not limited to, fun as a methodology of aesthetic engagement with the computational systems that attempts to draw out of them the unexpected, paradoxical, absurd and annoying, as in the case of JODI, or the pleasure of the DIY as with Arcangel. The fun of surf clubs thrives on utilizing or parasitizing upon the software gestures, interaction devices and computational habits of social media and other computational structures, allowing artistic strategies to be built that are constructed on radical relationality. Such relationality is productive of and takes into account idiocy and nonsense, memes and virality, where fun becomes a form of engagement with internet cultures, as in F.A.T. Lab projects.

A fun aesthetics also thrives on free and open-source software approaches, as otherwise material cannot be shared, changed and appropriated. This does not mean that these works cannot be found on commercial platforms and networks. For some practitioners, those are the places *par excellence*, where the image of purity and perfection that has attached itself to computer interfaces, software and computational networks can be uncovered and explored. All of the above draw on ridiculousness, frustration, alienation, lightness and estrangement. To summarize, a fun aesthetics can be seen as methodological, procedural, crafty, relational, cultural and conceptual. It does not shy away from the political or social concerns and advocates forms of openness that allow for various kinds of intervention, circulation and re-use. A fun aesthetics focuses on the investigation of the medium of computation and the internet, on the spaces and practices that are constantly changing and permanently moving. Fun, thus, enjoys a processual dynamics, both as a computational aesthetic form and as a collective practice.

There is much more to be said about the new generation of internet artists with regard to their aesthetics, their work with computational media and links to the earlier generation of internet artists and contemporary art in general. It is remarkable how well the second generation has found its way into art galleries and museum exhibitions. For building a proper understanding of this work, art critics and curators will have to move beyond conventional art aesthetics, which is too often about the form and look of a work, and start looking at how the project acts.[36] This means opening up to a wider social, economic, political and computational arena in complex ways. This also raises another question: can distributed, processual and software-based works be exhibited at all? The vernacular style and technique may remain untouched, but the energy, surprise, agony and illusion will disappear. What remains is just a form of aesthetics, without fun.

The presentation of *Highscreen* reminds us of the carelessness with which significant artworks are treated. Canonical artworks are dumped, soon to be forgotten. This is a humorous gesture, albeit with a very serious undertone. Revisiting older websites in the course of this chapter makes it abundantly clear how many of the links are now dead. Perhaps fun does not last forever, but it certainly shows its face when least expected, be it online or in the gutter. Trying to find it means fumbling round in the dark.

Notes

1 At the same time, as Josephine Bosma reminds us in her book *Nettitudes: Let's Talk Net Art*, Studies in Network Cultures (Rotterdam: NAi Publishers, 2011), the recycled computer has become a representation of media (art) activism, signifying digital divide.

2 In her chapter 'Early Conceptions of Humor: Varieties and Issues', in *The Psychology of Humor: Theoretical Perspectives and Empirical Issues*, Jeffrey H. Goldstein and Paul E. McGhee (eds) (New York: Academic Press, 1972), Patricia Keith-Spiegel created a typology of eight categories: (1) biological theories (e.g.

Darwin, 1872), which regard humour as an adaptive disposition; (2) superiority theories (e.g. Hobbes, 1968), which suggest that people laugh at others to whom they feel superior; (3) incongruity theories (e.g., Kant, 1951), which state that humour consists of incongruous events and situations; (4) surprise theories (e.g. Descartes, 1649), which suggest that humour requires suddenness and therefore weakens with repeated exposure; (5) ambivalence theories (e.g. Joubert, 1980), which posit humour as the result of opposing emotions or ideas within an appreciator; (6) release theories (e.g., Spencer, 1860), which regard humour as strain or stress relief; (7) configuration theories (e.g. Maier, 1932), which claim that humour depends directly on the resolution of incongruities; (8) psychoanalytic theories (e.g. Freud, 1928), which see humour as a result of economies of psychic energy that has been built up by and for repression. These can be broken down into three groups: the first three categories consider the function of humour; the second three focus on stimuli for humour; and the last two look at responses to humour. See also Veatch, Thomas C., 'A Theory of Humor', in *HUMOR, the International Journal of Humor Research* (Berlin: Mouton de Gruyter, 1998) 11(2): 161–215.

3 Mondrian, Piet, 'Neo-Plasticism: The General Principle of Plastic Equivalence', in *Art in Theory 1900–1990: An Anthology of Changing Ideas*, Charles Harrison and Paul Wood (eds) (Oxford: Blackwell Publishing Ltd, 2003, original 1920), 289–92.

4 The expression 'way to freedom' refers to the Futurist sense of 'words in freedom', where words were liberated from syntax, grammatology and typography. Marinetti, Filippo Tommaso, 'Geometric and Mechanical Splendour and the Numerical Sensibility', in *The Words and the Images. Text and Image in the Art of the Twentieth Century*, Jan Brand, Nicolette Gast and Robert-Jan Muller (eds) (Utrecht: Centraal Museum, 1991, original 1914), 43–6. For more information see among others Rasula, Jed and McCaffery, Steve (eds), *Imagining Language. An Anthology* (Boston: MIT Press, 1998).

5 Broos, Kees, 'Letter, Word, Text, Image', in *The Words and the Images. Text and Image in the Art of the Twentieth Century*, Jan Brand, Nicolette Gast and Robert-Jan Muller (eds) (Utrecht: Centraal Museum, 2001), 22–3.

6 Ibid., 129.

7 http://www.eai.org/title.htm?id=8784 (accessed 12.12.2013).

8 Other members are Paul B. Davis, Joe Beuckman and Joe Bonn, http://www.post-data.org/beige/ (accessed 12.12.2013).

9 Espenschied, Dragan, 'BEIGE. *Make World Festival at Lothringer 13'* (19 October 2001), http://www.post-data.org/beige/beige_make.html (accessed 12.12.2013).

10 '"Legacy Hackster", An Interview with Cory Arcangel', http://www.petitemort.org/issue01/02.shtml (accessed 12.12.2013).

11 A large emphasis on interactivity or participation can also turn people away. For a good analysis see Van Oenen, Gijs, 'Interpassivity Revisited: A Critical and Historical Reappraisal of Interpassive Phenomena', in *International Journal of Žižek Studies* 2(2) (2008).

12 Dawkins, Richard, *The Selfish Gene* (Oxford, Oxford University Press, 1976).

13 See, for example, Graham, Gordon, Genes: A Philosophical Inquiry (New York: Routledge: 2002).

14 Blackmore, Susan, *The Meme Machine* (Oxford and New York: Oxford University Press, 2011). For a critique of memetics and Blackmore, see Fuller, Matthew, *Media Ecologies. Materialist Energies in Art and Technoculture* (Cambridge, MA: MIT Press, 2005), 109–17, and Sampson, Tony, *Virality: Contagion Theory in the Age of Networks* (Minneapolis: University of Minnesota Press, 2012).

15 http://fffff.at/ (accessed 12.12.2013).

16 Possibly not coincidentally E.A.T's former address is only a few blocks away from the current Eyebeam Center where F.A.T. Lab originated.

17 Billy Klüver, E.A.T. Archive of published documents, http://www.fondation-langlois.org/html/e/page.php?NumPage=306 (accessed 12.12.2013).

18 See, for example, Yuill, Simon, 'All Problems of Notation Will be Solved by the Masses: Free Open Form Performance, Free/Libre Open Source Software, and Distributive Practice', in *FLOSS+Art*, Aymeric Mansoux and Marloes de Valk (eds) (Poitiers: GOTO10 and OpenMute, 2008), 64–91.

19 Dekker, Annet, 'Versions, E-mail Interview with Theo Watson' (Netherlands Media Art Institute, Amsterdam, 2009), http://nimk.nl/eng/versions-and-the-comment-as-medium (accessed 12.12.2013).

20 Cloninger, Curt, 'Commodify Your Consumption: Tactical Surfing /Wakes of Resistance', Black Mountain (2009), http://lab404.com/articles/commodify_your_consumption.pdf (accessed 12.12.2013).

21 For a discussion of idiocy, see Goriunova, Olga, 'New Media Idiocy', in *Convergence: The International Journal of Research into New Media Technologies*, (Sage, published online before print: 25 September 2012, 14 pp. Doi: 10.1177/1354856512457765 (accessed 12.12.2013).

22 http://fffff.at/about/ (accessed 12.12.2013). LOL as an acronym has several meanings, from look-a-like sites to laughing-out-loud. Inadvertently, in Dutch the word 'lol' translates as 'fun'.

23 An art work that comes to mind is *Form Art* (1997) by Alexei Shulgin, as well as his later *WIMP – the Windows Interface Manipulation Program* (2003) which plays with the icons and windows of the desktop. See http://easylife.org (accessed 12.12.2013).

24 This is pronounced in his darkly ironic online/offline performances *Arranged Online Performance* (2010–11), which take on the theme of a popular Chat roulette network. The performance series involves paid webcam services. First, Dullaart texts the webcam worker, paying a certain amount and consequently receiving the performance by the webcam worker, then he turns onto another webcam worker and positions the screens in such a way that the webcam workers perform to each other. Ultimately he confronts the performers with the audience by turning his webcam towards the audience.

25 See, among others, Troemel, Brad, 'From Clubs to Affinity: The Decentralization of Art on the Internet', in *Peer Pressure. Essays on the Internet by an Artist on the Internet*, ed. Brad Troemel (Brescia, Italy: LINK Editions, 2011), 33–45, and Olson, Marisa, 'Lost Not Found: The Circulation of Images in Digital Visual Culture', in *Words without Pictures*, ed. Alex Klein (New York: Aperture, 2008).

26 One could argue that these sites are exemplary of the 'surfing' mode which was once practised by many, before the search engines started personalizing search results.

27 This was for the exhibition *Versions* at the Netherlands Media Art Institute in Amsterdam which I curated together with Petra Heck and Constant Dullaart. A number of artists who rely, in their work, on creating a network of responses and comments, were invited to take part. We challenged them to temporarily exchange the internet for the static space of the gallery. Common threads of the exhibition were formed by questions about the significance of appropriation, authenticity and agency in the era of 'comment culture', http://nimk.nl/eng/versions (accessed 12.12.2013).

28 Thalmair, Franz, 'Re:Interview #015: Ever-Changing Chains of Work | Constant Dullaart', CONT3XT.net, http://cont3xt.net/blog/?p=4567 (accessed 12.12.2013).

29 Ibid.

30 Ibid.

31 See also Troemel, 'From Clubs to Affinity', 40.

32 See, for example, Dekker, Annet, 'Enabling the Future, or How to Survive FOREVER', in *A Companion to Digital Art*, ed. Christiane Paul (New York: Blackwell, 2014).

33 Guida, Cecilia, 'YouTube as a Subject: Interview with Constant Dullaart', in *Video Vortex Reader II Moving Images Beyond YouTube. INC Reader #6,* (eds) Geert Lovink and Rachel Somers Miles (Amsterdam: Institute of Network Cultures, 2011), 333.

34 See, for example, http://www.artfagcity.com/2008/09/29/lost-not-found-the-unnamed-sources/ (accessed 12.12.2013).

35 See, for example, Dekker, Annet, 'Pay with Your Privacy. A Conversation between Annet Dekker and Lernert & Sander' (2011), http://www.skor.nl/eng/projects/item/interview-lernert-sander (accessed 12.12.2013).

36 Especially with the earlier internet artists, often things that were said or claimed turned out to be a (strategic) hoax. As net artist Vuc Cosic says, '[T]here are tons of mystifications, starting with the origins of the name [net.art]', Bosma, *Nettitudes*, 163.

12

Material Imagination: On the Avant-gardes, Time and Computation

Olga Goriunova

The future is our only goal.

Varvara Stepanova and Alexander Rodchenko, Slogan from the newspaper *Art of the Commune*

Those who flick through the pages of old magazines for long enough know that each epoch had its own future, similar to the 'Future in the Past' of English grammar: as if the people of the past extend themselves into infinitude along a straight line, drawn through their own time at a tangent to eternity. Such a future never comes, because humankind walks into the future along a complex and barely comprehensible trajectory …[in these magazines] people from yesterday's tomorrow … stand in their pumped up space suits next to the chubby rockets, and above them an arrow of a space shuttle blasting off glides in the pale zenith – a painfully beautiful Noon of humankind …

Victor Pelevin, 'Soviet Requiem', in *Pineapple Water for the Beautiful Lady*[1]

delay itself is the pure form of time in which before and after coexist.

Gilles Deleuze, *Difference and Repetition*[2]

Introduction

This chapter explores the kinds of present time currently available and dominant as they are produced by the computational systems to be socially and culturally lived and, vice versa, imagined by art and culture to become and enter the computational streams. Such present times necessitate specific forms of relationality between the past and the future, between computational processes and the avant-gardes, offering variable versions of the future. Re-configured enactments of the present in its relation to the past and to the future are analysed in this chapter through a close reading of some of the computational devices of digital signal processing. I then attempt to take a discussion of the avant-gardes beyond the discourse of Utopia. I am particularly interested in the avant-garde as the provider of the multiplicities of the future, which I explore, among other examples, through some software art of the 1990s.

The starting point of the chapter is the sense of the loss of the future, the experience of the perpetual, expandable present. This kind of present results from changes in the mediation of time, and hence its construction, that have advanced beyond the 'end of history', and whose effects can be seen and are indeed understood through politics, analysis of financial instruments and trading algorithms or sociological methodologies.

For instance, Eyal Weizman maintains that there is a shift from the politics of justice to the politics of compassion, emitting 'the culture of immediate and direct action …. a culture of emergency [in the now]'.[3] 'The specter of the worst [totalitarianism] shapes the politics of the present', which then becomes a politics of the perpetual present, politics without a future.

'When utopia seems dangerous, what remains is only the pre-emptive management of … risk'.[4] Here, the 'perpetual economy of immanence'[5] lacks the redemptive mechanism that is 'in excess of all calculations' as thought by the first theorizer of the least possible evil, St Augustine.[6] In Weizman's interpretation of St Augustine, it can be maintained, it is the redemption that is the excess that used to provide a future.

Emma Uprichard explores the perpetual present as a 'sticky time' of the sociological imagination, which, through idiomatic methodological traps of computational methods and the abundance of digitally occurring, non-curated data, gets stuck in the present.[7] She says:

> The increased availability of real-time data sources increasingly means that 'to know' becomes not so much about how to predict the future, but how to predict 'now', or better still, to know about now before now has happened. … [W]here the 'past' and 'future' increasingly become a matter of hours or days, and ultimately more like our present 'present', the present itself becomes more

> and more plastic, to be stretched, manipulated, moulded and ultimately 'casted' by those who can access more of it in the supposed 'now'.[8]

And further: 'it is … possible to envisage a future of constant recursive presents, where we become stuck as we struggle to try to orientate ourselves according to redescriptions of the present as reconditioning the present'. Uprichard argues for becoming unstuck from a 'constant series of "presents"', from the plastic present, going beyond the 'stickiness' of time and temporality for a possibility of the futures to emerge.

The above are just two examples of the current conceptual unease in dealing with forms of time. Discussions of the avant-garde, on the other hand, have currently ceased their liveliness, as the avant-garde's claims on the future are circumscribed to the utopian and, thus, considered failed and dismissed as illusory constructions belonging to a rational and dominating modernity with its Saint-Simonian will to progress. While Baudrillard, Virilio and Bifo, each from different positions, have claimed that the present had either speeded up or became illusory to the point that the future vanishes,[9] such constructions call upon the conceptual power of both history, dealing with the past, and the avant-garde, self-named to initiate the future.

Charity Scribner writes:

> Until recently, utopian thought offered an alternative route away from implacable historical reality. Now, at the purported 'end of history' when time seems to pause in the eternal present, utopia veers into the longing for History itself … but also [the desire] to resuscitate the principle of hope that inspired much of the last century's social and cultural production.[10]

Indeed, various futures offered by the avant-gardes may be understood more productively and with sufficient nuance if premised not on the concept of Utopia, and understood as primarily a political claim, ultimately linking the Reason of Enlightenment to the calculation of the Cold War, or the dreamworlds produced in the West and East alike, as Buck-Morss writes, under the guise of high modernism or industrial modernity and thus eternally undermined with the departure of the 'faith into the modernizing process developed by the West'.[11] The genre of Utopia linking Plato's *Republic*, Thomas More's *Utopia* and Chernyshevky's *What Is to Be Done?* is much older than modernity itself and, beyond its importance for violent revolutions, is also profoundly connected to what Scribner, following Ernst Bloch's three-volume oeuvre on Utopia, called a 'principle of hope'.

In the same way that the discrediting of Reason does not halt thinking, connections between certain forms of the avant-garde, and especially the early Soviet avant-garde, with specific versions of Marxism, Hegelianism or the bleaker aspects of the Russian Revolution, must not obfuscate the recognition and recovery of the excessive, multimodal and luxurious reserve of potentiality offered by the avant-gardes. Buck-Morss specifically says, 'catastrophic effects need to be criticized in the name of the …utopian hope … not as a rejection of it'.[12]

We could usefully regard the principle of hope as the operation of hope, therefore focusing on the process that makes the avant-gardes and the constitution of the emergent futures. To distinguish the force of the avant-garde, itself a metaphenomenon not confined to art, from religion and post-secularism, sport or other spheres of activity, which Bloch considers when providing a comprehensive review of utopian thinking, I suggest calling the processes that sustain the avant-gardes and make them susceptible to the production of excessive plurality of the futures an operation of material imagination. The Russian word *mechta* means both dream and desire and harbours hope. The material imagination is not daydreaming, but a material work of imaginal production, that can lend itself to various kinds of capturing into amalgamative actualizations but whose potentiality is not used up by them.

The computational time of off-now

The avant-gardes used to be describable in terms of historical periods and conditions; they used to start and finish, and to recur. The concept of the avant-garde, by the mere morphological construction of the word used to signify it, prescribes a spatial relationship to time stretched through the body of the onlooker.

The organization of time into forms of spatialization has been questioned by Bergson, a theme subsequently developed by Deleuze, arguing for a continuum of duration, especially when the actualization of the virtual is concerned.[13] The chronological ordering of events of the duration of human life as intertwined with movement in space allowing for a different organization of a temporality has also been interestingly discussed by Haraway[14] who follows Bakhtin in his concept of chronotope, or timespace.[15] A chronotope, according to Bakhtin, organizes 'real actuality'.[16] The core to the chronotope is movement. Contrary to an understanding of such movement as only that of the (human) body in space is a proposal by Adrian Mackenzie to regard algorithmic space and time through, for instance, the movement of ordering and sequencing, via 'code constructs such as loops' and common data structures such as arrays.[17] Here, 'algorithms organize informatic time' and 'enliven movements lifted out of everyday and existing social orderings'.[18] The abstractions involved in the movement between computational input and output rely on topological transformation and re-ordering in time as 'concentration or intensification of movements'.[19]

Computational time-space is a social form of time, but there is what Mackenzie calls the 'crafting of computational time', which attunes software in order for it to be synchronized with everyday time. Computational chronotopes thus exist within a social framing, but through attempts to 'flow into everyday life',[20] through shifts, disjunctures and disconnections and, also, in the formation of topological strategies of continuity to align those different times.

Luciana Parisi writes about the urban design and topology of continuity in terms of '[anticipation of] the emergence of potential changes'[21] in which an

> '[a]lgorithmic mode of planning defined by an extended apparatus of prediction [is] able not only to establish the condition of the present through the retrieval of the past data, but also... change these conditions according to data variations immediately retrieved from the environment.[22]

The predictive calculation of spatio-temporal emergence that Parisi analyses through focusing on the invariable function in computational planning is a route to understanding specific kinds of computational crafting of space-time, with detailed attention to the evolving structures of the real time and to the contingent temporalities, among others.

The technicity of time and spatio-temporality, rigorously commented upon by Mackenzie and Parisi,[23] engenders a variety of technical mediations to generate ensembles which structure and construct lived time. The technical mediation of time here generates a regime 'from which time and space unfold'.[24] The computational mediation of time constituted in movements and operations afforded by delays, loops or predictive calculations flows into experienced time to produce new amalgams of time that structure lived reality by taking part in ensembles unfolding to produce it. Such an understanding of the production of lived reality draws upon a variety of scales: making new varieties of time available in a network of relationships between elements as artefacts (samples, pedals, data structures, units, code constructs, software gestures); as processes (sorting and sequencing, ordering and buffering, calculating and predicting, producing, playing back); the clashing of times; and as bodies mediated by computation effect the ensemble produced and the kind of time being dominant.

The computational mediation of time, unveiled by Mackenzie through the analysis of code constructs such as arrays or sorting, can further be usefully exemplified in software gestures dealing with music and noise as sonic realities existing through their unfolding in time. The sonic time and the time of computation here double, triple and quadruple: as sound is stored, computed, output, layered in with synthesized noise – such times zoom in and out of each other, overlay and intersect.

The idea of real time in computation is a useful starting point. Real time can mean that changes to the process can be introduced in the course of, say, a performance, or it can relate to the time when audio samples leave the computer,[25] a moment that is always behind logical time (that time corresponding to computing operations applied at irregular intervals performed prior to samples being output for real-time synthesis). The time of real-time composition – live performance – is real time, but the control and audio computations are carried out in turn prior to, and in the pockets in between, real time, according to the order of logical time.

> The reason for using logical time and not real time in computer music computations is to keep the calculations independent of the actual execution time of the computer which can vary for a variety of reasons, even for two seemingly identical calculations.[26]

So there is a real time and another real time running with delays and in parallel, and a buffer of different size and speed used for different purposes.

A multiplicity of the present time is actually a pre-digital idea:[27] naturally occurring echoes or a reverberation in the bathtub are very simple non-digital examples. Electronic technology, in turn, has been used to record, process and analyse music since the 1940s,[28] where most digital audio ‘effects’ have their predecessors. In general, as regards the computation of sound, the introduction of the personal computer cannot be seen as the unique starting point. Though there are some techniques which are only possible in the digital age, analogue synthesizers could do things to sound waves that digital synthesizers and computers carry on doing, and experimental work on early computers, mainly in universities in the late 1950s and 1960s, developed methods of audio analysis and synthesis that were promptly implemented in the commercial digital synthesizers of the late 1970s – a parallel line to the delayed development of desktop software that would later do the same and more.

Still, audio effects relied on mediation by some form of storage or slight delay. Before the advent of powerful computers, synchronized magnetic tapes were used as the form of storage and slowing one tape, or in the case of a loop, using reel-to-reel tape to make a tape loop were the mechanics behind such effects. This was often done during a performance and worked into the sound, a process that could be further complexified by adjusting the position of multiple recording and playback heads in studio conditions.[29]

With analogue electrical equipment, sound waves were transformed, via a transducer, into their electrical representation, which was then amplified and transformed back into the physical form via a loudspeaker. Work on sound happened between transducer and loudspeaker. Distortion, for example, could be achieved by increasing the electric power supply to an amplifier, altering the shape of audio waves coming through it. Filter effects changed the frequency spectrum of the wave: modulation split the audio signal in two and altered one while leaving the other unaltered in order to then join them together. Here, an electronic representation allowed for a sonic transformation to take place, and, with time-based effects, it relied on a time-delay inaudible to the human ear, on the deep present.

Until 1995, with the storage of audio signals becoming overwhelmingly digital, 'cutting, duplication, speed change and time reversal'[30] remained, but other techniques of synthesizing and altering sound, such as waveshaping, became more powerful – especially true in the case of those relying on the computation of complex mathematical formulae. Equally, techniques of additive synthesis (adding a few sound waves together) were explored, with some research and experimentation into amplitude modulation in the 1960s by Stockhausen, frequency modulation in the 1960s by John Chowning of Stanford University and waveshaping synthesis in the late 1960s by Jean-Claude Risset, and Yamaha, with its synthesizers, already making use of some of these techniques in 1975.[31] Still, some methods, such as wavetable synthesis and physical modelling, could only advance with digital signal processing technology and powerful computers.

Digital signal processing includes the delay (and buffer) as its fundamental data structure. It is the mediation of digital delay that is my key example of the computational crafting of new realities of time, as it is layered in with other kinds of time in relation to which things perform. Delay is when the signal is shifted by *d* samples; the time shift equals a delay of *d/r* time units, where *r* is the sample rate.[32] A digital delay line is a data structure that allows delay by a number of samples.

The digital delay line is the basis of many things in digital signal processing. Digital delay lines can be implemented as circular buffers (often used in data streaming) and are part of digital delay networks (which can be presented as time shifts or frequency domain alterations).[33] A delay line itself can change over time as in the case of a variable delay line. The delay line is the foundational structure of any filter since a delay network 'designed specifically for its frequency or phase response is called a filter'.[34] Furthermore, some application of filters can be used for Fourier analysis, resynthesis (additive synthesis) and subtractive synthesis. Digital delay lines are used in creating various kinds of audio effects and also in digital waveguide synthesis – a part of physical modelling synthesis first described by Avraamov as further introduced below.

The meaning of delays and buffers ranges from signifying elements in filter theory (such as in the digital delay line), being part of the process that produces sound, to referring to a form of computation that occurs in blocks (possibly outputting sound) to a rather more colloquial use, referencing the generic reliance of computational processes on timeshifts and redistribution. In this sense, they involve making copies of data, adding them together, going back to the calculation, storing the results up, storing the data as well as the computational process while it is in constant use in another location, and so on.

However different such uses are, the concepts and operations behind them take hold of time and make it work in rather new ways. With the digital delay line time runs unevenly. The previous slowing down and speeding up of sonic processes translate into the concentration and dilution of the present time. Present time becomes multiple: it slides apart to make space for the time of the calculation; real time expands; the delayed, the buffered is stored inside a pocket of the present, neither past, nor future, but what can be called *off-now*. The multiple present is seamful. Computational time adjusts through shifts and mismatches to the commonly held everyday present but, even below the 20 or 30 milliseconds inaudible to the human ear, sonic computational events transform the experiential construction of time, which lures humans into the macro and micro states of time, into the enduring forms of the present.[35]

To the detriment of the Bergsonian project, software-based functions continue spatializing time, grounding it in geometrical space and movement,[36] but they also draw out of time the capacity to take on new forms – to become pockets of time, to form loops in which the present is suspended, replayed, accumulated.

Goodman interprets Attali's idea of sound as the medium of the future by suggesting a future that is not near or immediately next, but the one that 'virtually coexists … [with] the past and opens up to its futurity'.[37] Such futurology of anticipation, of listening in to, of prophecy, gains power when it is supplemented with the concept of an active, material and excessive production of the future. The avant-gardes do not only hear the hidden societal heartbeat, but also constitute multiplicitous versions of the future. Crucially, such a temporality of the avant-gardes seems to enter into direct conflict with computationally mediated time with the ambition to produce novelty that ruptures the present temporality and leaps into the open and emergent future.

Despite Arendt's treatment of artistic work as instrumental, her concept of *natality*, an ability to initiate spontaneous novelty through action, in terms of the political understanding of freedom, can be usefully adapted to understand the futurity of avant-gardes.[38] In fact, it is the early Soviet avant-garde or the more recent avant-garde of certain digital art finding a revolutionary aesthetic in and across the blossoming of digital technology that might be seen to have produced aspects of the spontaneous freedom that Arendt is drawn to. The idea of fun that this volume pursues has something in common with the radical and spontaneous novelty that Arendt attributed to political action and I would like to think in relation to the (digital) avant-garde. Fun can be seen as related to the avant-garde's impetuous process of crafting the multiple futures. The initiation of the new constitutions of the future is a kind of fun that manifests or is disguised differently in every stratum in which it acts.

To the avant-gardes

The joint examination of computational time, avant-garde and the constitution of futures can offer a way of understanding the particular becoming of our digital reality. In fact, a claim I want to make here is that the avant-gardes operate within a different dynamic of unfolding wherein the avant-garde's future-in-the-past[39] becomes a means of the construction of our present and the future. The Future in the Past is a verb tense within English grammar that is used to describe what in the past was thought to be going to happen in the future. The future of the avant-garde, 'its only goal', according to Rodchenko and Stepanova, becomes stuck in the past, while forever remaining the future – it thus has it own form of existence and a relation to our past and present production and the production of our past, present and future. It is through the avant-garde's future-in-the-past that we have a range of futures, also a computational future, even if it currently seems to be happening all the time or not taking place at all.

A recent exhibition on the Russian sound technology pioneers of 1910–30s, curated by Andrei Smirnov and Liubov Pchelkina of the Theremin Centre, part of the Moscow Tchaikovsky Conservatory, makes a statement on the avant-garde's future-in-the-past succinctly clear: from the technology of ornamental sound proposed by Arseny Avraamov (1929–30), as popularized by the sound ornaments of Oskar Fischinger (1932), or Avraamov's conceptualizations and formulae for physical modelling synthesis, as well as descriptions of methods for additive synthesis as mentioned above, to the early versions of motion-tracking performed by Bernstein (1921–3), the innovations of the aesthetic revolutionaries of the beginning of the twentieth century had embarked on developing most of what could only be completed in the 1970s, 1980s or even 1990s.[40] More than that, it is the things that never led to further development and failed to complete that still retain the scintillating unattainability of art in the scale of their audacity in the creation of the novel material versions of the future – such as the 'Symphony of Factory Sirens' by Arseny Avraamov, a performance of 50 locomotive whistles, steamship blasts, hydroplane engines, factory sirens and two artillery batteries with machine guns and cannon for percussion. Not that technical innovation or technocratic idealization is glorified by such a statement; it is rather a methodology of making new, the fun of mediated cross-species and cross-agent communal creativity transfiguring the present that remains, in some form, in the future that can be found, however perversely modified, in today's creative industry policies, in the wishes of social network communities, in self-evolving art projects and in the plans of computationally mediated collective actions.

The avant-garde of early sound experimentation in the Soviet Union, among many other avant-gardes, strove to work according to a different logic, to offer a convergence between 'revolutionary imagination and material form',[41] to cut through the material, technological, political, social and subjective, to the very core, the engine of the making of the world, in order to reverse-engineer, disassemble and reassemble it, to make a new one, or better, a few. Whether offering an alternative version to capitalist modernity,[42] or 'bringing sensory form to utopian ideas',[43] such an excessive imaginary and material impulsion cannot be fully described through a closer engagement with society, power, technology or subjectivity; the engagement is with something more than society and politics, and beyond the technology at hand.

The avant-garde bites into the core of available means through which things become to create ways in which such processes of constitution can in turn become novel. The avant-garde is about assembling physical and metaphysical crossings at which being and becoming can be sustained in new ways. The avant-garde plugs into becoming before it is stratified and tries to bring about, not only a different actualization but a different mode of becoming. It does not only engage with the end-product, it rips open the process of its making in order to multiply, alter or affect the process and produce a materialist ontological revolution. Here, the aesthetics of the avant-garde participates directly in life. Both a reading of Nietzsche as a theorist of non-human-aimed art, of nature itself which has an art-state and a Guattarian notion of the aesthetic register that transforms and infects itself and other registers until they all function as aesthetic are useful orientations here.[44] Not-just-art, larger than art, non-art, the avant-garde is not a political or social project any more than it appears as an aesthetic current. It goes beyond; as an inherent revolution in the manner of its actualization, it cannot fail.

To go back to the beginning of this section, the fact that the avant-garde exists as a vector directed at the future in relation to its present and seems to have failed to become present when its future passes does not mean that it is utopian. It means that it establishes a unique relation to time and to technology.

Future-in-the-past

Deleuze, through Bergson, argued that the present only exists through and by entering into a relation with its own past. 'The present is not',[45] and it is through the past that the present is constructed. The past becomes the virtual through which every present, of which there are multiple, actualizes or becomes through differentiation. The present time is a duration, a becoming: it actualizes from the virtual along multiple lines (one or more of which can be stratification).

The 'past co-exists with its own present on various levels'[46] and 'at the same time'.[47] The present never ceases and goes back to itself as a past, which is here an 'ontological memory that is capable of serving as the foundation of the unfolding of time'.[48]

The present of the avant-garde, which is the future, is the line of actualization of becoming, which stays open. It is not a form of cosmic memory, an inspiration, a shared ideology, a shared past. If such lines and manners are regarded as a future, it is only because there is a becoming involved, but it also implies that they cannot fail, because they are not circumscribed by the now. Time does not check on them whether they became, have failed to become or have become past. Their present is the pure manner of becoming, of differentiation, and such a present always remains a future even if it is in the past.

In the section on the novella in *A Thousand Plateaus* Deleuze and Guattari write, 'They have a future but no becoming'.[49] The future-in-the-past, by contrast, offers myriad becomings, but poses a question of what the future is. Early sound pioneers relate to something that will happen, which does not become a failed Utopia, a rigid trauma, but remains multiplicitous becomings. The present never exists because it immediately passes and becomes past; but the future cannot pass, especially when it is the future-in-the-past that never took place. With the avant-garde's time, in the form of something that happened and avoided happening, we have a past that remains a future, that is becoming, dynamic and multiple. The future-in-the-past is always present, unlike the future, thus, in a certain way it is the only real future. It does not wither, it is a future at hand, a way to become, to be, a line, a modality, a technique, a device, a methodology. Such sets of manners of actualization cannot be described exclusively as creative techniques to be exploited and subsumed. They offer futures, and of a sort that is still to be understood.

Buffered avant-gardes of the 1990s

It has often been argued that the conversion of avant-garde into mainstream no longer takes decades, and its recursion is rapid. There are many little avant-gardes, with scarce time to claim failure or success. When the material of culture is processed and advances through calculation, it acquires computational characteristics. A recursion or interslicing of the avant-gardes as they are computed is made prone to algorithms similar to those behind time-based effects: avant-gardes can be delayed to form an echo, played back and looped, sampled, filtered, stored, mixed with a live feed, synthesized or predictively modelled. Visions realized, enquiries still pertaining to the future, past breakthroughs, a mixture of times are processed to be operated upon, among other, through the architectural-conceptual element of the delay, of the buffer. It is tempting to say that the novelty of the open futures, of manners of actualization is buffered; the future-in-the-past takes the form of the off-now, forming an immediate space, multiple real time, being stored, delayed, layered, fed back.

Software art and net art, a pair of futures of the past two decades, offered, as avant-gardes do, material imaginaries of human-technical networks to form novel realities. Shaping the roles of culture and art in relation to computational systems and vice versa, they conceptualized software agencies, tested its manifestations, imagination, habits; they described its social and political futures, its breakages. Rather than offering solutions, software art and its precursor – net art, commented on in other chapters of this volume – developed ways of thinking computation that revealed its increasingly central position in the processing of culture, society and subjectivity.

It is through projects such as *Auto-Illustrator*,[50] a vector graphic design application by Adrian Ward of Signwave that jointly won the Transmediale 2001 artistic software award, that software ceased to be considered to be something contained within certain boundaries and acquired messiness, multiplicities in action, coupling human labour with other agents, including those within and outside the running code. When *Auto-Illustrator* comments 'It's boring. Choose another tool' or offers advice on the fanciness of colour used, it does not only make work somewhat 'fun' but realizes, in material terms, the condition of culture subjected to computational processes, a human confronted with his/her own technical subjectification, the condition of technicity that has entered a renewed cycle of human genesis. Software art worked through and exploited software materiality, programmable creativity, multi-species distributions of agency, the conditionality of acts driven by the design of computational networks. All in all, if software art was not of foundational importance to software, as it existed in its massified form, it certainly became a form of material imagination that brought the art and culture of the 1990s and early 2000s closer to the realization of, and acting upon as well as creating, its own conditions.

A list of artists making Google Art, who were quickly hired by Google (one would be Douwe Osinga)[51] or of software gestures subsequently implemented in major software applications (the wave-like movement of Mac OS X interface elements following a decade of 'digital folklore' projects breaking up the stability of WIMP interface design)[52] might well be a good inventory to assemble, not least for the cause of registering prior art – as such things become patented as part of a 'look and feel' design. Ideas and gestures that were appropriated, and that proliferated, could be discussed in detail, starting from those under the general header of 'forerunners' to social networking, blogs and the aesthetics of connection. Here Eva Wohlgemuth and Kathy Rae Huffman's report on *The Siberian Deal*, essentially a text and image travel diary with a feedback option, HTML-ed and uploaded to the server (through FTP) (1995)[53] could be mentioned alongside the *Refresh* project (1996), a painstakingly broken dream of decentralization, as if precipitating the development of RSS feed technology[54] and Heath Bunting's semi-hoax CCTV project (1997) literally realized 13 years later.[55] To pause for a while here, Bunting's parodic invitation to monitor four webcam streams and report anything suspicious to local police via a form is now programmed into a piece of software available from the company Internet Eyes, which aims to use an 'army of volunteers' to watch and interpret CCTV footage online.[56] Anti-utopian parody bypasses the original mixing times. Here, the technologies of digital signal processing, data compression, cameras, scenarios for interaction, fibre optics, interface design and others are all combined in the imagining and construction of a social user, a model of communication, a manner of spending time, a way to be as society. Here, sets of software operations are as important as their active participation in the construction of a user, and of those middle territories formed of people and networks, watching, processing the stream. Here, net art and software art come up with ideas for the culture to be, with ways for it to become: devices and gestures on the rise, marking scheduled human-technical updates.

The readiness and promptness with which the ideas thought up in certain currents of net art and exploratory art practices of the 1990s were embedded into commercial platforms and applications align these art movements with that of Russian Constructivism. As in the Russian Constructivism, with its focus on 'real technical construction' and the invention of the 'productivist artist-engineer', where drawings were also to function as technical blueprints, currents in net art and software art invented conceptual and computational gestures, personae and forms of communal actions in 'productivist' manners, either somewhat close to those of a technical plan, developed, played out with and put forth or as actual technical objects. Indeed, net art explored the computational undergirding of life coming about through computational means, with works acting as software functions, even if only imaginary, but ready to be reprogrammed and included in large software packages – similar to

Constructivism producing its projects in the spirit of integration into everyday societal lives. Constructivism seeking new societal structures through the work of material imagination can be seen as aligned with the computational art of the 1990s and 2000s co-constructing and inhabiting the Temporary Autonomous Zones and the zones beyond. Here, not only can the visual language of new media be seen as descending from the avant-garde, as famously analysed by Lev Manovich,[57] but also the aesthetico-political impetus. Furthermore what is shared is a sense of failure and stuffiness of reaction following suit (Stalinist nomenclature, in the case of Soviet avant-garde, and a neoliberal death dance of biopolitical co-option, in net art and software art). The unhelpful bitterness in the face of this almost immediate co-option of the avant-gardistic endeavours is another reason to try and think beyond this model, drawing upon the leaking and excessive operation of hope, of material imagination and its temporal configurations.

Certainly, it is not only the projects that became the (often commercial) present that matter. Things that did not become, that still remain a future, are even more important. As an example, the instruments and imagination of the *Web Stalker* browser by the I/O/D group (1997)[58] conceptualized as a means of enquiry into the material structures of networks, of introducing perspectivalism in relation to the Web, such as in the 'Crawler' and 'Map' function, remain rather unimplemented, outside a domain of specialist software. The 'Crawler' crawls the target URL, proceeding to follow all the HTML pages that they link to, whereas the 'Map' visualizes the relation of pages within the website and to outside links. It has been used outside of art scenes, for instance, to track all child links and find hidden pages; something being taken further with Web crawlers and network analysis and visualization techniques applied to data obtained from social networking platforms. The future software functions here are the gestures of materially imagining a mode of technical being that feeds on engagement with computational infrastructures rendered thick and dense in the acts of using them which become one with modes of understanding them or indeed of producing such modes of being.

A combination of what became reality but ceases to pass and what remains a future but is presently neatly at hand, in relation to the avant-garde of old technical art forms, characterizes the contemporary landscape. The examples given refer to ideas mainly: concepts that call computational formulations into being, problems that necessitate software development. Such a claim can be seen to be one of the many ways in which this volume tries to reformulate the relationship between computation and the history of concepts, programming and social sciences, software development and art. The fun of the avant-garde, stored up in the future-in-the-past or enlisted by predominant data structures, fills in the techno-human formulations of today as they unfold and roll over to institute tomorrow. Not only is the importance of avant-garde's fun to be noticed here, so also is the changing shape of the novel actualizations of reality that they establish. If a computational mediation of the delay and buffer takes hold of the future-in-the-past looped in the multiple stretchy presents, new sets of problems and actions can be drawn: ones that institute breakages and discontinuity and are able to work through the present in order to draw out multiplicitous futures, destabilizing the delay, folding the loop.

There are still formulations, events and expressions that become past. A delayed future-in-the-past is joined by the once present and then passed, such as the celebrated democracy of HTML, a channel for everyone to raise their voice and make a statement. These means of enunciation are currently rendered useless if they are unsupported by the complications of contemporary web design and scripting or conversely if they are embraced in (mainly corporately owned) blogs and CMSs. In a similar vein, Zittrain warns of the dangers of the 'appliancization' of the internet, a condition in which the once indubitable open architecture of the Web and desktop computing is jeopardized and may effectively be closed into a ramified system of single-service appliances, granting little control and no place for manoeuvre in relation to what they are, can become or do to the user and the network.[59]

Past simple and past perfect, present, future in the past and future enter into a new interlocking machine, an agreement expressed most acutely in the form of computational systems and one that must be recognized if the becoming is not to be lost to just very few options in the face of technological temporal complexity. The software-based processing of time creates layers inside layers, well-managed or glitchy delays, parallel, intersecting and closed-circuited time structures. It is not only the enduring ephemeral described by Chun, but also a buffered present that cannot become either past or produce a future. Present and threatened, withered away, gone, left in limbo, multiplicitous futures form clusters whose temporal dimensions are confusing.

Multiplicitous futures and the incomputable machine

Can there be the versions of the future that are not utopian? Can there become a present that fulfils the avant-gardes' multiplicitous futures? Is computational time always the continuous present? Perhaps, in fact, the answer to all these questions should be negative. The vector of the future is not a temporal movement due to arrive at a state of completion. The avant-gardes are not reducible to the cause of an action, whether immediate or delayed. And the computability is never complete, but contains the incomputable.

Luciana Parisi comments on the computational construction of the future and Massumi's work in the following way: 'a preemptive mode of power foreclosing futurity into actualities is not the same as the incomputable machine of the event'.[60] She argues that topological invariants in computation that pre-empty change through continuous variation and '[re-program] the event before it can happen' are not the only existent computational reality, but, drawing from Whitehead, she suggests that the transduction of qualities into quantities is

> infected with abstract non-denumerable relations of pure quantities, eternal objects: discrete yet permanent relations adding novel character to existing... relations… Each parametric extensive relation is hosting another order of quantities that cannot be contained by the number of its actual members.[61]

These are: 'abstract quantitative order of relations'[62] always escaping overall continuity; the dark spaces of 'parasitic quantities', discontinuity, contingency, the incomputable always included in the computational processing of matter. For Parisi, there is always, grounded in the logic and computational elements, the problematic and contingent that exceed the control and exhaust pre-emption. The computed continuous present includes the incomputable, the rupturing black holes of other times.

The other times that can burst through the buffered elastic present include those with the potentiality of the avant-gardes. The avant-gardes offer distinctive and intensive kinds of production of the multiplicitous becomings, the imaginary crafting of the futures. Such processes, such futures in the form of becomings, the multiple unfoldings into and of the future are not one, are in excess. As an excess, such reserve of material imagination also escapes buffering as much as it is subjected to it. The avant-garde's multiplicities of the future, inclusive of multiple histories, still exist in the ruptures of today as an enactive force that cannot be fully lost to the perpetual present.

The material working out of multiplicitous futures by the avant-gardes is not a dream that 'comes against reality and fails to insert itself',[63] but an operation of imagination. The force of such material imagination is not non-rigorous or intrinsically linked to domination, and the call upon its intensity is not nostalgic. It is indeed the avant-gardes' multiplicitous futures that are the reserve and excess to be called upon when the construction of the multiple present is required, whether or not such a present (and the future) is to be posthuman, ecologically sustainable, transdisciplinary and ethical.[64]

In *Author and* Hero *in Aesthetic Reality* Bakhtin wrote that the future cancels out the present and the past, rather than following from them.[65] What expression such an actualization will take, we shall see.

Notes

1 (Moscow: Eksmo, 2011).
2 (London: Continuum, 2008: 152).
3 Weizman, Eyal, *The Least of All Possible Evils. Humanitarian Violence from Arendt to Gaza* (New York and London: Verso, 2011), 37.
4 Ibid.
5 Ibid., 21.
6 Ibid., 38.
7 Uprichard, Emma, 'Being Stuck in (Live) Time: The Sticky Sociological Imagination', in *The Sociological Review*, 60 (2012): S1,124–38, doi: 10.1111/j.1467-954X.2012.02120.x.
8 Ibid., 133.
9 Virilio, Paul, 'The Visual Crash', in *Ctrl[Space]: Rhethorics of Surveillance from Benthan to Big Brother* (Cambridge, MA: MIT Press, 2002); Baudrillard, Jean, *The Vital Illusion* (New York: Columbia University Press, 2000); Berardi, Franco 'Bifo', *After the Future* (Oakland, CA: AK Press, 2011).
10 Scribner, Charity, *Requiem for Communism* (Cambridge, MA: MIT Press, 2005), 9–10.
11 Buck-Morss, Susan, *Dreamworld and Catastrophe. The Passing of Mass Utopia in East and West* (Cambridge, MA: MIT Press, 2002), x.
12 Ibid., xiv.
13 Deleuze, Gilles, *Bergsonism* (New York: Zone Books, 1991); Bergson, Henri, *Matter and Memory* (New York: Zone Books, 1991).
14 Haraway, Donna J., *Modest_Witness@Second_Millenium.*

FemaleMan©_Meets_OncoMouse. Feminism and Technoscience (New York and London: Routledge, 1997), 41.

15 Bakhtin, Mikhail, *The Forms of Time and Chronotope in the Novel*, Complete Works, Vol. 3, *Theory of the Novel (1930–1961)* (Moscow, 2012, in Russian) [in Russian, *Formy vremeni i khronotopa v romane*].

16 Bakhtin, *The Forms of Time and Chronotope in the Novel,* 341–2.

17 Mackenzie, Adrian, *Cutting Code. Software and Sociality* (New York: Peter Lang, 2006), 51–7.

18 Mackenzie, *Cutting Code,* 56, 57.

19 Ibid., 57.

20 Ibid., 64.

21 Parisi, Luciana, 'Digital Design and Topological Control', in 'Topologies of Culture' (special issue), (eds) Celia Lury, Luciana Parisi and Tiziana Terranova, *Theory, Culture & Society*, 29(4 and 5) (2012), 167.

22 Ibid., 167.

23 Mackenzie, Adrian, *Transductions. Bodies and Machines at Speed* (London and New York: Continuum, 2002), 89, and Parisi, 'Digital Design and Topological Control'.

24 Mackenzie, *Transductions*, 95.

25 Puckette, Miller, *The Theory and Technique of Electronic Music* (London: World Scientific, 2010), 61.

26 Ibid.

27 Goodman, Steve, 'The Sonic Algorithm', in *Software Studies: A Lexicon*, ed. Matthew Fuller (Cambridge, MA: MIT Press, 2005), 230.

28 Puckette, *The Theory and Technique of Electronic Music,* xiii.

29 Gorokhov, Andrei, *Muzprosvet* [in Russian], http://www.muzprosvet.ru (accessed 01.07.2013).

30 Puckette, *The Theory and Technique of Electronic Music,* 27.

31 Miranda, Eduardo Reck, *Computer Sound Design. Synthesis Techniques and Programming* (Oxford: Focal Press, 2002).

32 Puckette, *The Theory and Technique of Electronic Music,* 181.

33 A delay network can be worked with in two ways – in the time domain ('we draw waveforms as functions of time, and consider delays as time shifts') and the frequency domain ('we dose the input with a complex sinusoid (the output is a sinusoid of the same frequency) and report the altitude and/or phase change as a function of frequency' (Puckette, *The Theory and Technique of Electronic Music,* 185). The time and frequency domains are two ways of understanding delay networks, depending, again, on time: when delays are short and humans cannot hear them, the frequency domain picture is used.

34 Puckette, *The Theory and Technique of Electronic Music,* 225.

35 Wendy Chun's concept of an 'enduring ephemeral' can be seen as related here. Wendy Hui Kyong Chun, 'The Enduring Ephemeral, or the Future is a Memory', in *Place Studies in Art, Media, Science and Technology. Historical Investigations on the Sites and the Migration of Knowledge*, (eds) Andreas Broeckmann and Gunalan Nadarajan (Weimar: Verlag und Datenbank für Geisteswissenschaften, VDG, 2009). See also the concept of microsound developed by Curtis Roads.

36 Goodman, Steve, 'Timeline (Sonic)', in *Software Studies: A Lexicon*, ed. Matthew Fuller (Cambridge, MA: MIT Press).

37 Goodman, Steve, *Sonic Warfare. Sound, Affect, and the Ecology of Fear* (Cambridge, MA: MIT Press, 2010), 49.

38 Arendt, Hannah, *The Human Condition* (Chicago: University of Chicago Press, 1998), 8, 9, 247.

39 For example, 'I thought I would get that job though I had a feeling that the interview was going to be a disaster'. Both 'would get' and 'was going to be' here are in Future in the Past tense.

40 Theremin Centre, http://theremin.ru (accessed 09.05.2012). Smirnov, Andrey, *Sound in Z. Experiments in Sound and Electronic Music in Early 20th Century Russia* (London: Koenig Books, 2013).

41 Buck-Morss, *Dreamworld and Catastrophe*, 64. Buck-Morss offers her own theory of time/temporality of the avant-garde.

42 Kiaer, Christina, *Imagine No Possessions. The Socialist Objects of Russian Constructivism* (Cambridge, MA: MIT Press, 2005), 1.

43 Buck-Morss, *Dreamworld and Catastrophe*, 65–6.

44 Nietzsche, Friedrich, *The Birth of Tragedy and Other Writings,* trans. Ronald Speirs (Cambridge: Cambridge University Press, 1999); Guattari, Félix, *Chaosmosis*, *an Ethico-Aesthetic Paradigm* (Sydney: Power Publications, 1995).

45 Deleuze, *Bergsonism* (Cambridhge, MA: MIT Press, 1991) 55.

46 Ibid., 74.

47 Ibid., 61.

48 Ibid., 59.

49 Deleuze, Gilles and Guattari, Félix, *A Thousand Plateaus, Capitalism and Schizophrenia* vol. 2 (New York: Continuum, 2004), 216.

50 Signwave *Auto-Illustrator* 1.2, http://swai.signwave.co.uk (accessed 09.05.2012).

51 Some of Douwe Osinga'swork can be accessed through the Runme software art repository by searching for his name at: http://runme.org (accessed 09.05.2012).

52 See Goriunova, Olga, *Art Platforms and Cultural Production on the Internet* (New York and London: Routledge, 2012).

53 Wohlgemuth, Eva and Huffman, Kathy Rae, *The Siberian Deal* (1995), http://www.t0.or.at/~siberian (no longer available).

54 *Refresh* online project (1996), no longer available online.

55 Irational, *CCTV* (1997), http://www.irational.org/heath/cctv/ (accessed 09.05.2012).

56 *BBC News*, 'Cash Prizes for Catching CCTV Criminals' (04.12.2009), http://news.bbc.co.uk/1/hi/technology/8393602.stm (accessed 09.05.2012).

57 Manovich, Lev, *The Language of New Media* (Cambridge, MA: MIT Press, 2002).

58 I/O/D *Web Stalker* (1996), http://bak.spc.org/iod/ (accessed 09.05.2012).

59 Zittrain, Jonathan, *The Future of the Internet and How to Stop It* (New York: Penguin Books, 2008).

60 Parisi, 'Digital Design and Topological Control', 185.

61 Ibid.

62 Ibid., 186.

63 Foucault, Michel, *The Birth of Biopolitics, Lectures at the College de France 1978–1979* (New York and London: Palgrave, 2010), 320.

64 Braidotti, Rosi, *The Posthuman* (New York and London: Polity, 2013).

65 Bakhtin, Mikhail, 'Author as Hero in Aesthetic Activity', in Russian, in *Ehstetika Slovesnogo Tvorchestva* (Moscow: Iskusstvo, 1979), 107.

Notes on Contributors

Christian Ulrik Andersen is Associate Professor in Digital Aesthetics and Culture at Aarhus University, Denmark. His research addresses interface criticism in diverse contexts including games, cities and business platforms. He has published the book *Interface Criticism* (co-edited with Søren Pold) (Aarhus University Press, 2011), and he is also the co-editor of a series of peer-reviewed newspapers published in collaboration with reSource transmedial Culture (Berlin) as well as the journal *A Peer-reviewed Journal About*, aprja.net (with Geoff Cox).

Wendy Hui Kyong Chun is Professor of Modern Culture and Media at Brown University, USA. She has studied both Systems Design Engineering and English Literature, which she combines and mutates in her work on digital media. She is author of *Control and Freedom: Power and Paranoia in the Age of Fiber Optics* (MIT Press, 2006), and *Programmed Visions: Software and Memory* (MIT Press, 2011). She is editor of numerous books and special journal issues including *New Media, Old Media* (Routledge, 2005). Her current book project is: *Habitual New Media*.

Geoff Cox is Associate Professor in the Department of Aesthetics and Communication, and Participatory IT Research Centre, Aarhus University, Denmark. He is also Adjunct Faculty of Transart Institute (Germany/USA), and part of the self-institution Museum of Ordure. His research interests lie in the areas of contemporary art and performance, software studies, network culture and a reappraisal of the concept of publicness. His publications include *Speaking Code: Coding as Aesthetic and Political Expression* (MIT Press, 2012), written with Alex McLean. He is also an editor for the DATA browser book series published by Autonomedia, where he co-edited *Economising Culture* (2004), *Engineering Culture* (2005), *Creating Insecurity* (2009) and *Disrupting Business* (2013).

Annet Dekker is an independent researcher, curator and writer. Currently she is core tutor at Piet Zwart Institute, Rotterdam. Previously she was co-producer of Funware at MU in Eindhoven, worked as Web curator for SKOR (2010–12), was programme manager at Virtueel Platform (2008–10), head of exhibitions, education and artists-in-residence at the Netherlands Media Art Institute (1999–2008), and editor of several publications on digital art and issues of preservation. In 2008 she began a PhD on strategies for documenting net art at the Centre for Cultural Studies, Goldsmiths, University of London.

M. Beatrice Fazi is a researcher, writer and lecturer. She is now completing a PhD at the Centre for Cultural Studies at Goldsmiths, University of London, with a thesis on the aesthetics of computation. This thesis explores the centrality of abstract entities and processes to the construction of experience. Her present research investigates the limits of formal reasoning in relation to computation, and addresses the ways in which these limits shape the ontology of computation vis-à-vis cultural and scientific notions of incompleteness and incomputability.

Matthew Fuller's books include *Media Ecologies, Materialist Energies in Art and Technoculture* (MIT Press, 2005), *Behind the Blip, Essays on the Culture of Software* (Automedia, 2002) and *Elephant & Castle* (Autonomedia, 2012). He is co-author with Usman Haque of *Urban Versioning System v1.0* (ALNY, 2011), and with Andrew Goffey of *Evil Media* (MIT Press, 2012). He is editor of *Software Studies: A Lexicon* (MIT Press, 2008) and a co-editor of the journal *Computational Culture*. He works at the Digital Culture Unit, Centre for Cultural Studies, Goldsmiths, University of London. http://www.spc.org/fuller/

Andrew Goffey is Associate Professor in Critical Theory and Cultural Studies at the University of Nottingham. He is co-author with Matthew Fuller of *Evil Media* (MIT Press, 2012), co-editor with Eric Alliez of *The Guattari Effect* (Continuum, 2011) and co-editor with Roland Faber of *The Allure of Things* (Bloomsbury, 2014). A member of the editorial collective of *Computational Culture*, his research explores the connections between philosophy, science and culture, with a particular emphasis on software.

Olga Goriunova is Assistant Professor at the Centre for Interdisciplinary Methodologies, University of Warwick. She is the author of *Art Platforms and Cultural Production on the Internet* (Routledge, 2012) and co-editor of a number of volumes, including *Readme: On Software Art and Cultures* (Aarhus University Press, 2005). Her research focuses on digital culture and cultural theory, aesthetics, computation and nonsense. She is a co-editor of the journal *Computational Culture*.

Brigitte Kaltenbacher completed her PhD in 2009 at Goldsmiths, University of London. Her research interests focus on implicit cognitive processes, e.g.

subconscious learning, motivations and their connection to creativity. Her background is in information design, which she applies to the latest digital interaction technology. She has over 15 years of professional experience in the area of digital interaction including convergent/mobile internet solutions, IP-TV, multimedia and TV production and currently works as commercial User Experience researcher in London.

Andrew Lison is an Andrew W. Mellon Graduate Fellow in the Department of Modern Culture and Media at Brown University, USA. He is the co-editor (with Timothy S. Brown) of *The Global Sixties in Sound and Vision: Media, Counterculture, Revolt* (Palgrave Macmillan, 2014). His work has appeared in *New Formations* and *Science Fiction Studies* and is concerned with the relationship between technological development, aesthetic practice and radical politics from modernity to the present, with an emphasis on digital media.

Adrian Mackenzie is Professor in Technological Cultures, Department of Sociology, Lancaster University. He has published work on technology: *Transductions: Bodies and Machines at Speed* (Continuum, 2002–6); *Cutting Code: Software and Sociality* (Peter Lang, 2006); and *Wirelessness: Radical Empiricism in Network Cultures* (MIT Press, 2010). He is currently working on the circulation of data intensive methods across science, government and business in network media. He co-directs the Centre for Science Studies, Lancaster University.

Alex McLean (aka Yaxu) is a musician and researcher based in Yorkshire. He is one-third of the live coding band Slub (with Dave Griffiths and Adrian Ward) and co-founder of the TOPLAP live coding organization. He coordinates the Algorave promoters and ChordPunch record label, both promoting algorithmic music, and has organized around 80 Dorkbot electronic art events in Sheffield and London. Alex is currently Research Fellow in Human/Technology Interface and Deputy Director of ICSRiM, University of Leeds. He completed his PhD thesis 'Artist-Programmers and Programming Languages for the Arts' in 2011 at Goldsmiths, University of London.

Michael Murtaugh is a freelance programmer and member of the Constant Art and Media Association in Brussels. He has taught on the Piet Zwart (Networked) Media Design Programme in Rotterdam since 2003.

Luciana Parisi is a Reader in Cultural Studies and runs the PhD Programme at the Centre for Cultural Studies at Goldsmiths, University of London. In 2004, she published *Abstract Sex: Philosophy, Biotechnology and the Mutations of Desire* (Continuum Press). Her research looks at the asymmetric relationship between science and philosophy, aesthetics and culture, technology and politics to investigate potential conditions for ontological and epistemological change. She has recently published a monograph *Contagious Architecture. Computation, Aesthetics and Space* (MIT Press, 2013).

Simon Yuill is an artist, writer and programmer. His work includes the use of interview and research processes, film, publishing and custom software systems. He was the inaugural winner of the Vilém Flusser Theory Award (Berlin, 2008), has been a Research Resident at the Piet Zwart Institute (Rotterdam, 2005), Institute for Advanced Studies Visiting Fellow at the University of Warwick (2013) and Honorary Visiting Research Fellow with the Centre for Cultural Studies at Goldsmiths College (London, 2011–14).

Index