LEA is a publication of Leonardo/ISAST.

LEONARDO
THE INTERNATIONAL SOCIETY FOR THE ARTS, SCIENCES AND TECHNOLOGY

Sabancı Üniversitesi

Goldsmiths
UNIVERSITY OF LONDON

VOL 17 NO 2 VOLUME EDITORS LANFRANCO ACETI AND SIMON PENNY
CONTRIBUTORS STEPHANIE BOLUK, MAURO CARASSAI, KENNY CHOW, SHARON DANIEL, KRISTEN GALVIN, FOX HARRELL, SNEHA VEERAGOUDAR HARRELL, GARNET HERTZ, JI-HOON FELIX KIM, PATRICK LEMIEUX, ELISABETH LOSH, MARK MARINO, MICHAEL MATEAS, CHANDLER B. MCWILLIAMS, CARRIE NOLAND, ANNE SULLIVAN, NOAH WARDRIP-FRUIN, JICHEN ZHU

SPECIAL ISSUE

DAC 09

after media : embodiment and context

Copyright 2012 ISAST
Leonardo Electronic Almanac
Volume 17 Issue 2
January 2012
ISSN: 1071-4391
ISBN: 978-1906897-16-1
The ISBN is provided by Goldsmiths, University of London

LEA PUBLISHING & SUBSCRIPTION INFORMATION

Editor in Chief
Lanfranco Aceti lanfranco.aceti@leoalmanac.org

Co-Editor
Özden Şahin ozden.sahin@leoalmanac.org

Managing Editor
John Francescutti john.francescutti@leoalmanac.org

Art Director
Deniz Cem Önduygu deniz.onduygu@leoalmanac.org

Graphic Designer
Zeynep Özel

Editorial Assistant
Ebru Sürek

Editors
Andrea Ackerman, Martin John Callanan, Connor Graham, Jeremy Hight

Editorial Board
Peter J. Bentley, Ezequiel Di Paolo, Ernest Edmonds, Felice Frankel, Gabriella Giannachi, Gary Hall, Craig Harris, Sibel Irzık, Marina Jirotka, Beau Lotto, Roger Malina, Terrence Masson, Jon McCormack, Mark Nash, Sally Jane Norman, Christiane Paul, Simon Penny, Jane Prophet, Jeffrey Shaw, William Uricchio

Contributing Editors
Nina Czegledy, Susan Collins, Leonardo Da Vinci, Anna Dumitriu, Vince Dziekan, Darko Fritz, Marco Gillies, Davin Heckman, Saoirse Higgins, Jeremy Hight, Denisa Kera, Frieder Nake, Vinoba Vinayagamoorthy

Editorial Address
Leonardo Electronic Almanac
Sabanci University, Orhanli - Tuzla, 34956
Istanbul, Turkey

Email
info@leoalmanac.org

Web
» www.leoalmanac.org
» www.twitter.com/LEA_twitts
» www.flickr.com/photos/lea_gallery
» www.facebook.com/pages/Leonardo-Electronic-Almanac/209156896252

Copyright © 2012
Leonardo, the International Society for the Arts, Sciences and Technology

Leonardo Electronic Almanac is published by:
Leonardo/ISAST
211 Sutter Street, suite 501
San Francisco, CA 94108
USA

Leonardo Electronic Almanac (LEA) is a project of Leonardo/The International Society for the Arts, Sciences and Technology. For more information about Leonardo/ISAST's publications and programs, see http://www.leonardo.info or contact isast@leonardo.info.

Leonardo Electronic Almanac is produced by Passero Productions.

Reposting of this journal is prohibited without permission of Leonardo/ISAST, except for the posting of news and events listings which have been independently received.

The individual articles included in the issue are © 2012 ISAST.

Making Inroads: Promoting Quality and Excellency of Contemporary Digital Cultural Practices and Interdisciplinarity

I would like to welcome you to the first special volume of the Leonardo Electronic Almanac. *DAC09: After Media: Embodiment and Context*, is a volume that generated from the conference by the same name that Prof. Penny chaired at the end of 2009.

DAC09: After Media: Embodiment and Context is the first of a series of special volumes of the Leonardo Electronic Almanac that are realized in collaboration with international academic, editors and authors.

Prof. Penny was inspired for this LEA special issue by the continuous developments in the interdisciplinary arena and in the fields of new media and digital art culture. He wanted to collate research papers that would provide the seeds for innovative thinking and new research directions. The authors featured in this volume, to whom we are most grateful for their hard work, will provide the reader with the opportunity to understand and imagine future developments in the fields of digital art culture and interdisciplinarity.

As I look at the electronic file of what we now internally refer to simply as *DAC09* the first issue of the revamped LEA, *Mish Mash*, printed and delivered by Amazon, sits on the desk next to my keyboard. The possibilities and opportunities of e-publishing, which also has physically printed outcomes, provide me with further thoughts on the importance and necessity of the work that is done by 'small publishers' in the academic field. The promising news of a new open access journal to be launched by The Wellcome Trust or the 'revolution' of researchers against Elsevier through the website http://thecostofknowledge.com/ with 9510 Researchers Taking a Stand (Thursday, April 12, 2012 at 10:57 AM) highlights the problems and issues that the industry faces and the struggles of young researchers and academics.

The contemporary academic publishing industry has come a long way from the first attempts at e-publishing and the revolution, if it can be defined as such, has benefited some and harmed others.

As the struggle continues between open access and copyrighted ownership,[1] the 'revelation' of a lucrative academic publishing industry, of economies of scales, of academics exploited by a system put in place by publishing giants (into which some universities around the globe have bought into in order to have an internationally recognized ranking system) and the publishers' system of exploitation structured to increase the share of free academic content to then be re-sold, raises some essential questions on academic activity and its outputs.

ISSN 1071-4391 ISBN 978-1-906897-16-1

The answers to these problems can perhaps be found in the creativity of the individuals who participate in what is, at times, an harrowing process of revisions, changes, reviews, replies and rebuttals. This is a process that is managed by academics who donate their time to generate alternatives to a system based on the exploitation of content producers. For these reasons I wish to thank Prof. Simon Penny and all the authors who have contributed to *DAC09: After Media: Embodiment and Context*.

Simon Penny in his introduction to this first LEA special volume clearly states a) the importance of the *DAC09* and b) the gravitas and professional profile of the contributors. These are two points that I can support wholeheartedly, knowing intimately the amount of work that this volume has required in order to maintain the high standards set by *Mish Mash* and the good reception it received.

For this reason in announcing and presenting this first special volume I am proud to offer readers the possibility of engaging with the work of professionals who are contributing to redefining the roles, structures and semantics of new media, digital art practices and interdisciplinarity, as well as attempting to clarify what digital creativity is today and what it may become in the future.

The field of new media (which are no longer so new and so young – I guess they could be better described as middle aged, slightly plump and balding) and digital practices (historical and contemporary) require new definitions and new engagements that move away from and explore beyond traditional structures and proven interdisciplinary partnerships.

DAC09: After Media: Embodiment and Context is a volume that, by collating papers presented at the DAC09 conference, chaired by Prof. Simon Penny, is also providing recent innovative perspectives and planting seeds of new thinking that will redefine conceptualizations and practices, both academic and artistic.

It also offers to the reader the possibility of engaging with solid interdisciplinary practices, in a moment in which I believe interdisciplinarity and creative practices are moving away from old structures and definitions, particularly in the fraught relationship between artistic and scientific disciplines. If 'cognitive sciences' is a representation of interdisciplinarity between artificial intelligence, neurobiology and psychology, it is also an example of interdisciplinary interactions of relatively closely related fields. The real problem in interdisciplinary and crossdisciplinary studies is that these fields are hampered by the methodological problems that still today contrapose in an hierarchical structure scientific methodologies versus art and humanities based approaches to knowledge.

This volume is the first of the special issues published by LEA and its appearance coincides with the newly revamped website. It will benefit from a stronger level of advocacy and publicity since LEA has continued to further strengthen its use of social platforms, in fulfillment of its mission of advocacy of projects at the

ISSN 1071-4391 ISBN 978-1-906897-16-1

intersection of art, science and technology. *DAC09* will be widely distributed across social networks as open access knowledge in PDF format, as well as being available on Amazon.

I extend a great thank you to all of the contributors of *DAC09: After Media: Embodiment and Context* and wish them all the very best in their future artistic and academic endeavors.

Lanfranco Aceti
Editor in Chief, Leonardo Electronic Almanac
Director, Kasa Gallery

ACKNOWLEDGEMENTS

I would like to thank Ozden Sahin, LEA Co-Editor, for having delivered with constancy another project of which LEA could be proud. The LEA special issues are more similar to small books – 200 pages is not a small endeavor – that require special care and attentive selection.

I am very grateful to Prof. Simon Penny for the hard work that he has put into this volume and to the authors who have patiently worked with us.

To all of you my heartfelt thanks.

DAC09: After Media: Embodiment and Context is the first special volume of the Leonardo Electronic Almanac to be followed by many others that are currently in different stages of production, each of them addressing a special theme and focusing on bringing to the mainstream of the academic debate new forms of thinking, challenging traditional perspectives and methodologies not solely in the debates related to contemporary digital culture but also in the way in which these debates are disseminated and made public.

To propose a special volume please see the guidelines webpage at: http://www.leoalmanac.org/lea-special-issues-submission-instructions/

REFERENCES AND NOTES

1. Thomas Lin, "Mathematicians Organize Boycott of a Publisher," *The New York Times*, February 13, 2012, http://www.nytimes.com/2012/02/14/science/researchers-boycott-elsevier-journal-publisher.html (accessed March 20, 2012).

 ISSN 1071-4391 ISBN 978-1-906897-16-1

Two decades of Digital Art and Culture

An introduction to the LEA DAC09 special edition

by

Simon Penny

Director of DAC09
Professor of Arts and Engineering
University of California Irvine

This volume of LEA is composed of contributions drawn from participants in the 2009 Digital Art and Culture conference held at the University of California, Irvine in December 2009. DAC09 was the eighth in the Digital Art and Culture conference series, the first being in 1998. The DAC conference series is internationally recognized for its progressive interdisciplinarity, its intellectual rigor and its responsiveness to emerging practices and trends. As director of DAC09 it was these qualities that I aimed to foster at the conference.

The title of the event: After Media: Embodiment and Context, was conceived to draw attention to aspects of digital arts discourse which I believe are of central concern to contemporary Digital Cultural Practices. "After Media' queries the value of the term 'Media Arts' – a designation which in my opinion not only erroneously presents the practice as one concerned predominantly with manipulating 'media', but also leaves the question of what constitutes a medium in this context uninterrogated. 'Embodiment and Context' reconnects the realm of the digital with the larger social and physical world.

'Embodiment' asserts the phenomenological reality of the fundamentally embodied nature of our being, and its importance as the ground-reference for digital practices. 'Embodiment' is deployed not only with respect to the biological, but also with reference to material instantiations of world-views and values in technologies, a key example being the largely uninterrogated Cartesianisms and Platonisms which populate computational discourse. Such concerns are addressed in contemporary cognitive science, anthropology and other fields which attend to the realities of the physical dimensions of cognition and culture.

'Context' emphasises the realities of cultural, historical, geographical and gender-related specificities. 'Context' brings together site-specificity of cultural practices, the understandings of situated cognition and practices in locative media. The re-emergence of concerns with such locative and material specificity within the Digital Cultures community is foregrounded in such DAC09 Themes as Software and Platform Studies and Embodiment and Performativity.

The DAC09 conference included around 100 papers by an international array of contributors. In a desire to be maximally responsive to current trends, the conference was to some extent an exercise in self-organisation by the DAC09 community. The call for papers and the structure of the event was organized around nine conference themes which were themselves the result of a call to the community for conference themes. The selected themes were managed largely by those who proposed them. Much credit for the success of the event therefore goes to these hard-working 'Theme Leaders' : Nell Tenhaaf, Melanie Baljko, Kim Sawchuk, Marc Böhlen, Jeremy Douglass, Noah Wardrip-Fruin, Andrea Polli, Cynthia Beth Rubin, Nina Czegledy, Fox Harrell, Susanna Paasonen, Jordan Crandall, Ulrik Ekman, Mark Hansen, Terry Harpold, Lisbeth Klastrup, and Susana Tosca, and also to the Event Organisers: David Familian, Michael Dessen, Chris Dobrian, Mark Marino and Jessica Pressman. I am particularly grateful to Ward Smith, Information Systems Manager for DAC09, who for two years, as my sole colleague on the project, managed electronic communications, web design and the review and paper submission processes amid, as he would put it, a 'parade of indignities'. In the several months of final planning and preparation for the event, the acumen and commitment of Elizabeth Losh and Sean Voisen was invaluable.

I first published on what we now refer to as digital arts in 1987. [1] Not long after, I was lucky enough to have the opportunity to attend the first ISEA conference in 1988. Since that date I have been actively involved in supporting the development of critical discourses in the field, as a writer, an editor and an organizer of events. My role as director of the DAC09 conference gave me a perspective from which to reflect on the state of digital arts discourse and its development over two decades. As I discussed in a recent paper, [2] the first decade on media art theory was a cacophonous interdisciplinary period in which commentators from diverse fields and disciplines brought their expertise to bear on their perceived subject. This created a scenario not unlike that of various viewers looking into a house via various windows, none of them perceiving the layout of the house, nor the contents of the other rooms. In the ensuing decade, a very necessary reconciliation of various disciplinary perspectives has occurred as the field has become truly a 'field'.

While post structuralist stalwarts such as Deleuze and Derrida continue to be referenced in much of the more critical-theory oriented work in Digital Cultures, and the condition of the posthuman and posthumanist are constantly referenced, theoretical reference points for the field are usefully broadening. The emerging field of Science and Technology Studies has brought valuable new perspectives to media arts discourses, counterbalancing the excesses of techno-utopianism and the sometimes abstruse intellectualism of post-structuralist theoretical discourses. In this volume, Mark Tuters provides an exemplar of this approach in his *Forget Psychogeography: Locative Media as Cosmopolitics*, bringing Rancière and Latour to bear on a discussion of HCI, Tactical Media and Locative Media practices. Tuters provides a nuanced argument replete with examples which questions the sometimes, superficial and dogmatic re-citation of the originary role of the Situationists with respect to such practices. At DAC09, Connor McGarrigle also took a thoughtful revisionist position with respect to the Situationists. [3]

In this context, the new areas of Software Studies and Platform Studies have emerged and have been nurtured in previous DAC conferences. In this spirit, Chandler McWilliams attempt to "thread the needle between a reading of code-as-text that obfuscates the procedural nature of code, and an overly technical description of programming that reinstates the machine as the essential arbiter of authentic acts of programming" is emblematic of the emergence of Software Studies discourses which are quintessentially interdisciplinary and erudite on both sides of the science wars divide. Similarly, Mark Marino's meditations on heteronormativity of code and the Anna Kournikova worm call for what he calls Critical Code Studies, here informed by queer theory. In their proposal for an 'AI Hermenteutic Network' Zhu and Harrell address the question of intentionality, a familiar theme in AI critical discourse (i.e., John Searle 'Minds,

ISSN 1071-4391 ISBN 978-1-906897-16-1
ISSN 1071-4391 ISBN 978-1-906897-16-1

Brains and Programs' 1980). Citing Latour, Agre, Hayles and others, they offer another example of the science-wars-sidestepping technical development based in interdisciplinary scholarship noted in the discussion of Chandler McWilliams' contribution.

Another trend indicative of the maturation of this field is its (re)-connection with philosophical discourse. In this context, the deep analysis of Electronic Literature in terms of Wittgensteinian Language Games by Mauro Carassia is something of a tour de force. While a tendency to extropianism is here not explicitly discouraged, this discussion places such technological practices squarely as indicators of transition to post-human subjectivity, and in the process, open the discussion to phenomenological, enactive and situated critiques as well a drawing in the relevance of pre-cognitivist cybernetic theorisation.

One of the aspects of contemporary media arts discourse which I hoped to foreground at DAC09 was questions of embodiment and engagement with contemporary post-cognitivist cognitive science. Several papers in the current collection reflect such concerns, and indeed they were foregrounded in several conference themes. One example of the value of the application of such theory is evidenced in Kenny Chow and Fox Harrells leveraging of contemporary neuroscience and cognitive linguistics in their deployment of the concept of "material-based imagination" in their discussion of Interactive Digital Artworks. In a quite different approach to embodiment and computation, Carrie Noland discusses choreography and particularly the choreography of Cunningham, with reference to Mauss and Leroi-Gourhan, and with respect to digital choreographic tools.

The DAC community did not choose to make Game Culture a focal theme in DAC09 – perhaps because the field has grown so quickly and has built up a structure of conferences and journals. Nonetheless, gaming culture was referenced throughout the event, and was the subject of numerous presentations, such as Josh and Karen Tannenbaums reconsideration of 'agency as commitment to meaning', which addressed the acknowledged problematic of the tension between authorial and user agency in terms of a critique of the humanist subject. Like wise, phraseology such as Boluk/Lemieux's: "player performance in and around games has matured to the point of beginning to express underlying serial logics through heavily mannered gameplay mechanics" (in their contribution to this volume) signals the establishment of a mature and erudite critical theory of games and gaming. On a more technical note, Sullivan/WardripFruin/Mateas make an argument for enriching computer game play by application of artificial intelligence techniques to the authoring of 'quests'.

As Digital Arts became established as a practice the question of pedagogy inevitably arose – what to teach and how to teach it. Though rhetorics of convergence pretend to the contrary, one cannot dispute the profound epistemological and ontological dilemmas involved in attempting to bring together intellectual environments of such disparate communities as engineers, artists and critical theorists, in the classroom and the lab. Interdisciplinarity was therefore the ground upon which these programs were developed, and each context inflected that idea with its own color. My own reflections on the subject are published at *Convergence.* [4] It therefore seemed timely to address pedagogy at DAC09. In the process of elaboration of digital cultural practices, such emerging practices have themselves come into consideration as pedagogical tools and systems. In this volume, Elizabeth Losh surveys and discusses various pedagogical initiatives (mostly in Southern California) deploying digital tools and environments. In a contribution which crosses between the pedagogy thematic and concerns with cognition, Harrell and Veeragoudar Harrell offer a report on a science, technology, engineering, and mathematics (STEM) educational initiative among at-risk students which considers the relationships between users and their virtual identities.

In his essay, Garnet Hertz discusses the work of three artists – Reed Ghazala, Natalie Jeremijenko, and Tom Jennings. None of them 'media artists' in the conventional sense, they, in different ways and for different purposes, re-purpose digital technologies. Rounding out this volume is presentation of two online artworks by Sharon Daniels which were presented at DAC09. *Public Secrets* and *Blood Sugar* are elegant web-based art-works, both poetic and examples of a committed activist practice.

In my opinion, this collection offers readers a survey of fields addressed at DAC09, and an indication key areas of active growth in the field. Most of them display the kind of rigorous interdisciplinarity I regard as characteristic of the best work in the field. While the science-wars rage on in certain quarters, in media arts discourse there appears to be an attitude of intelligent resolution – a result in no small measure of the fact that a great many such commentators and theorists have taken the trouble to be trained, study and practice on both sides of the great divide of the 'two cultures', and to take the next necessary step of attempting to reconciling or negotiate ontologies traditionally at odds. This professional profile was very evident at DAC09 and is represented by many of the contributors in this volume. Such interdisciplinary pursuits are in my opinion, extremely intellectually demanding. The obvious danger in such work is of superficial understandings, or worse, a simple re-citation of a new canon of interdisciplinary media studies. Dangers that, happily, none of the papers grouped here, and few of the papers presented at DAC09, fell victim of. ■

The electronic proceedings of DAC09 are available at this link: http://escholarship.org/uc/ace_dac09

REFERENCES AND NOTES

1. "Simulation Digitization, Interaction: The impact of computing on the arts," *Artlink, Art+ Tech Special Issue* 7, no. 3 and 4 (1987).
2. "Desire for Virtual Space: the Technological Imaginary in 90s Media Art," in *Space and Desire. Scenographic Strategies in Theatre, Art and Media*, eds. Thea Brejezk et al. (ZHdK Zurich: Zurich University of the Arts, 2010).
3. This paper, and all DAC09 papers referenced here, are available as part of the DAC09 proceedings, online at http://escholarship.org/uc/ace_dac09 (accessed March 2010).
4. Simon Penny, "Rigorous Interdisciplinary Pedagogy: Five Years of ACE," *Convergence* 15, no. 1 (February 2009): 31 - 54.

ISSN 1071-4391 ISBN 978-1-906897-16-1
ISSN 1071-4391 ISBN 978-1-906897-16-1

Leonardo Electronic Almanac

Volume 17 Issue 2

ISSN 1071-4391 ISBN 978-1-906897-16-1
ISSN 1071-4391 ISBN 978-1-906897-16-1

Language in The Other Software

by

Chandler B. McWilliams

A B S T R A C T

Friedrich Kittler's analysis of software in his essay "There is no Software" evacuates the programmer from the realm of the computer *by focusing too intently on the machine and its specific, material existence. As a result, he posits the material action of computers, in the form of voltages, as the essential site of the being of computers. This paper attempts to thread the needle between a reading of code-as-text that obfuscates the procedural nature of code, and an overly technical description of programming that reinstates the machine as the essential arbiter of authentic acts of programming. By reasserting the presence of the programmer and exploring the variety of types of coding, this essay offers an alternate description of the being of software, one which emphasizes not just the execution of code on the machine, but also the programmer's role as reader and writer of code.*

 ISSN 1071-4391 ISBN 978-1-906897-16-1

INTRODUCTION

Friedrich Kittler's essay "There is no Software" argues for a narrowly defined, materialist conception of authentic uses of circuit-based computational machines. To this end, he dismisses software and high-level code languages, along with architectural abstractions inherent in the design of most computer systems, as unnecessary obfuscations which hide the nature of the machine itself. Furthermore, he deploys a conception of code-as-text the writing of which can be understood in terms familiar to the writing of natural language. These two ideas together necessarily leave little room for the programmer in the creation of computational artifacts. Reasserting the presence of the coder opens the door for a new understanding of code-as-text without falling into the trap of reading code as literature or a machinic essentialism of voltages.

NOT EXACTLY WRITING

Programming language, writing code; it seems clear that we are talking about language in the everyday sense, about texts that are written. [1] Software is of course written in a language, so why not bring to bear the mature theories of text on the world of software? "There is no Software," like many texts in software and critical code studies, talks frequently of writing. But the word writing offers up an irresistible temptation to talk about text. Writing code is writing like writing music is writing. This is not to say that music theory is the place to find insights into software, but to emphasize the multiple flexible meanings of the word writing. Music can be written sitting at a piano, with clicks on the screen, or with symbols on paper. Text may or may not be involved. Writing music is about manipulating sound. Few would argue that theories of text are relevant to understanding music. We should be similarly wary of over-identifing code with written text. Just as writing music is about manipulating sound, not symbols, writing software is about manipulating procedures, not language.

The flip side of understanding code-as-text is a conception of reading that places the machine at the center of the act of programming. Reading becomes reading-by-the-machine. Source code is compiled and turned into assembly, then translated into opcodes, which eventually become voltages, the final, true language of the machine. It is voltages, after all, that integrated circuits traffic in. "All code operations, despite such metaphoric faculties as call or return, come down to absolutely local string manipulations, that is, I am afraid, to signifiers of voltage differences." [2]

Even the binary codes we're told so much about are an abstraction on top of these voltages, 1 is just a name for five volts, and 0 is a name for ground. [3] Combined with literary theory's concern for what our writing does (a concern shared by Kittler), reading-by-the-machine offers a clear answer for software:

ISSN 1071-4391 ISBN 978-1-906897-16-1

writing code produces a synchronized choreography of voltages in integrated circuits.

It is all too easy to collapse the entire process and thus to do away with software. If code is always and only concerned with the eventual manipulation of voltages, then in a sense, what does it matter? Why this long process? We know what our writing does: it (eventually) affects voltage differentials in a silicon chip. Wouldn't it be simpler to slough off these abstractions and get as close to the machine as possible?

We are rightly struck by the mystery of the written word. The magic of not being able to map phrases in natural language [4] to specific behaviors or states in the mind of the reader is enticing. The wonderful ambiguity of language.

Writing, in Western culture, automatically dictates that we place ourselves in the virtual space of self-representation and reduplication; since writing refers not to a thing, but to speech, a work of language only advances more deeply into the intangible density of the mirror, calls forth the double of this already doubled writing, discovers in this way a possible and impossible infinity, ceaselessly strives after speech, maintains it beyond the death which condemns it, and frees a murmuring stream. This presence of repeated speech in writing undeniably gives to what we call a work of language an ontological status unknown in those cultures where the act of writing designates the thing itself, in its proper and visible body, stubbornly inaccessible to time. [5]

Is this then the status of writing software? Writing which designates the thing (voltages) in itself? Continuing this line of thought, the ambiguity of natural language is lost on the machine; there is only source code to assembly to opcodes to voltages. Read as a one-to-one mapping; the meaning and action is understood, explained, fixed. If this is the reading to compliment the writing of software, then the writer of code is no more than an operator manipulating a machine to produce a specific predetermined result. Like a switchboard operator, meticulously and uncreatively plugging circuits.

NOT EXACTLY VOLTAGES

When code is understood as literature, and so the techniques and questions of literary theory are asked of it, the machine will always emerge as the final answer. Missing from these discussions are the writer, the coder, the programmer. Rejecting the notion that code operates as a literary text allows us to reassert the presence of the coder in the code.

But if not text, then what? To avoid the temptations of the terms text, writing, and literature, let us say that code is an artifact. Specifically an artifact for describing and designing procedures and systems. [6] Code comes in many languages. Unlike the babel of human languages, programming languages tend to differ depending on how quickly the software can be coded, how easily the code is to write (there is a tacit relationship between the difficulty of writing the code and the speed with which the code will run), or on what hardware the software needs to run. The key differentiating factors however are the assumptions inherent in each language about how the coder thinks. This last and most important feature of a programming language is perhaps the primary reason for the development of new languages, methodologies, and cognitive styles. A consequence of this computational babel effect is that there is not one ideal language for writing a given piece of software. The choice of language is far more influenced by the skillset and needs of the coder than properties of the software being coded. There is rarely if ever one clear, best choice. What's more, it is not always possible to identify which requirements of the software will play a central determining role in making a choice of language until the coding has already begun. Thus the practice of prototyping and sketching solutions in a familiar environment to gain a better understanding of the problem itself. Here is one similarity with the writing of a text; the process of coding itself is often the only way to gain an understanding of what is to be coded.

An over-emphasis on the voltages in the silicon cannot account for the multiplicity of programming languages. The same piece of software, if written in a different language, will become a different set of voltages when compiled and executed. And since there are no clear machine-centric metrics by which to judge one set of opcodes as superior to another – judgements about speed, memory efficiency, etc. are all relative to the purposes of the human user – then how can these voltages serve as the ultimate measure of what it is that our (software) writing does? It cannot, and it fails to do so because writing code is not about manipulating voltages any more than writing music is about manipulating vibrations.

Processes and systems are the core concern of code. Writing software is writing procedures; defining rules and bounds for action, creating the possibility for behavior, form, and interaction. From the simplest utility script to the most realistic physics simulator, the common thread is the code which describes a particular process for achieving a task. There are of course, varying degrees of open-endedness and complexity across these two examples. A script that converts file names to lower case will, if written well, perform reliably and always produce the expected outcome. A simulation like those in video games is less straightforward. The outcome is not always predictable. This allows for a unique and rewarding gaming experience when played repeatedly, but also opens the door for simulation as a tool to model and eventually predict the outcome of the interaction of massive numbers of variables and processes. This can perhaps be most clearly stated in terms of how code operates in the arts. Here generative and parametric processes create the possibility of form; code creates a world of possibility within constraints rather than a particular form.

WHERE IS CODE EXECUTED?

A piece of software could always have been written in a different language yet perform the same task; often performing that task in a different way, with different voltages in the machine, and different mental models in the coder. Loosening the relationship between code and the machine lets us ask the question: Where is code executed? By Kittler's account, only in the machine after its eventual conversion to voltages; the programmer is only an operator tasked with controlling the machine. But if programming languages often perform the same task differently, offering important differences only to the programmer, then what do these differences tell us about programming? The keys ways in which languages differ is in terms of the mental models they offer and the assumptions they make about how code should be written. The multiplicity of ways of thinking about software indicates that code must to some extent be run in the mind of the programmer. Run with far less speed, complexity, and precision, but executed nevertheless. How else would programming be possible? The extent to which we can recognize that a programmer knows what effect a line of code will have is precisely the extent to which we can recognize that she has already run a simulation of that code in her head. The only other option is that coding is just smashing together symbols which are sent to the machine with fingers crossed. So software must have an effect on how the programmer conceptualizes a problem – in the form of mental

ISSN 1071-4391 ISBN 978-1-906897-16-1
ISSN 1071-4391 ISBN 978-1-906897-16-1

models and cognitive styles – and also exists as a description of a procedure interpretable by other coders without the intervention of the machine, without ever becoming voltages. To speak of software only in terms of voltages is no more interesting that to discuss a painting only in terms of the electro-chemical activity in the brain of the painter.

OBFUSCATION / ABSTRACTION

Kittler takes pains to describe the layers operating behind software. High-level programming languages are turned into assembly which then becomes opcodes and eventually voltages in the circuitry of the machine. This parallels another layering, an application (WordPerfect in his example) running over an operating system, which is in turn running on top of BIOS. He characterizes this as obfuscation, a complicated series of frauds perpetrated in the name of preserving (creating) intellectual property. However this layering is not so simple. Take the example of a driver and car. The car has pedals and a wheel which hide the underlying details of how it moves. The driver rarely needs to know if the car is powered using gas or electricity, he just needs to know that pushing the long vertical pedal makes it go. In programming terms we might say that the interface of the car (pedals and wheel) hides the implementation (engine, transmission, etc). Much like the abstractions of software, these abstractions are often useful, but can effectively put the user at the mercy of the designers of the system in question. Just as it is increasingly difficult to repair a new car in a home garage, the unavailability of the source code for popular software packages make them nearly impossible to alter.

However, there are other purposes for these abstractions and seeming obfuscations. For one, higher-level programming languages are "higher" in that their syntax and organization does not directly parallel the opcodes required by the machine. This makes them easier to learn and to think with. One can imagine a continuum between machine language and human language. Along this line, "higher" simply means a few steps closer to the human and away from the machine. Without these abstractions, programming would likely still be something of a dark art performed by self-appointed wizards at well-funded universities. But as it is we have visual languages, scripting languages, languages for artists, and languages for children. To write code for the machine always requires a change in our thought; points on this continuum never fully reach the human. It is always a meeting somewhere in-between: a becoming-machine of the programmer. This becoming cannot be summarized with phrases like "think like a machine." It is instead a thinking-along-with the machine. A direct engagement with the structures and potentials of a particular machine running a particular piece of code. It is here that code differs from other process-oriented languages, the most pervasive of which are legal codes. The law turns on interpretation of language and precedent; the meaning and application of legal documents evolve over time. Software codes on the other hand do not afford such ambiguity. The play and flexibility in software operates at the level of the processes being written, not at the level of language. Insofar as one uses a standardized language ANSI C, Java, etc. there is the assumption of standard execution, something unique to computation, and something which obscures the obverse side of procedural thought.

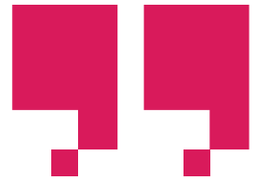
Quoting The Waite Group's Macroassembler Bible, Kittler tells us that "BIOS services hide the details of controlling the underlying hardware from your program." [7] This hiding is not necessarily malicious. Because programming requires concurrent reading and executing of code in the mind of the programmer, increases in complexity of the software necessarily bring increases in the difficulty of the mental execution. Rather than always-already indicative of a patronizing concealment, this "hiding" is often a useful tool to allow a complex system to be modularized and thus thought-through. There is only so much one can hold in one's head at one time. This process of encapsulation thus allows the coder to trust that a certain element will behave as advertised and therefore put out of her mind until she needs to change the behavior or the element does something unexpected. The heart needn't worry how the liver works as long as it keeps working.

EXPRESSIVE EXECUTION

Understanding that code is also for people, that it is always executed to some extent in the mind of the coder, opens the door for expression at the level of the text of the code. If the machine is placed at the center of the human-machine assemblage that is computer programming, we lose the ability to make judgements about the code at the level of text precisely because all questions about the value of one snippet of code versus another are settled by how the code runs on the machine. Thinking of code as something that must be intelligible for others, or more often for oneself in the future, lets us engage the broad flexibility of style and methodology present in all programming practices.

The simplest example of this is the choice of variable names. Variables in code store bits of data, they are in a sense the "nouns" of programming. Most programming languages give the coder relative freedom to name variables, for example many currently popular languages only require that variable names begin with a letter of the alphabet or the "_" symbol, and then contain only letters, numbers, or the "_". From the machine's perspective, it makes no difference how a variable is named; it could be given any valid name and the code would function the same way. As the programmer though, the choice of a name can make a significant difference in the legibility of the program. Just because a variable could be named something else doesn't mean it makes no difference what it is named. In fact this simple subject is at the center of countless ongoing battle about the proper way variables should be named. Countless systems have evolved over the years to discipline coders to follow certain naming rules "for their own good." The most famous of these systems is the so-called Hungarian Notation developed by Charles Simonyi in the 1970s

Thinking of code as something that must be intelligible for others, or more often for oneself in the future, lets us engage the broad flexibility of style and methodology present in all programming practices.



ISSN 1071-4391 ISBN 978-1-906897-16-1
ISSN 1071-4391 ISBN 978-1-906897-16-1

in which the coder includes a mnemonic for the type of data the variable represents in the variable's name often leading to unpronounceable and hard to read names.

Beyond the world of idioms and best-practices, variable, class, function, and method naming acts as a site for expressing the intentions of the coder, or introducing metaphors to represent what the code is doing. The possibilities of terse, literal, playful, etc. all contribute to how the code is conjured into being in the mind of the programmer often regardless of that code being technically executed in the machine. Here code-as-text and code-as-process are merged; it is far simpler to imagine and model a complex system if the elements of that system have reasonable linguistic handles to hold on to. In this simple act of naming entire ontological systems are woven.

Increasingly code is used as a medium for artistic expression in both the visual and plastic arts and literature. In addition to pieces of art created with computers, pieces of source code itself are distributed as pieces in their own right. Work such as Zach Blas' "transCoder: Queer Programming Anti- language" uses the syntax of code, an API to be exact, to critique the heteronormative structures at play in technology. This is also seen in Perl poetry, where poems are written or transcribed into the computer language Perl. Both of these uses ask us to consider the necessity for a piece of code to successfully execute on a machine. Looking at numerous Perl poems for instance, one is met with pieces that range from syntactically valid but missing key pieces that would allow them to actually run, to programs that run and express and idea not necessarily present in the execution of the code itself, to other pieces that produce truly stunning output while the code itself operates as a beautiful piece of writing. All of the points on this continuum tell us something about the nature of code. Socio-political API's that end at defining the names and descriptions of possible functions of mythical systems of liberation reexamine the dominant language of control in our time, code, and explore how that language could be put to other purposes. Source code that treats the syntax of code as just one among many literary styles continues this exploration, while making use of in-jokes present in each language and forces ambiguities between syntax and expression. And finally executable codeworks take the source language itself as a realm of play; pushing creative freedom and expression to the limit within the constraints of executability given by the machine.

HARDWARE ESSENTIALISM

Integrated circuits, the hardware at the core of all digital computers, require strictly defined paths for electrons to travel through on the chip. Without these controls, the chip would, more often than not, do nothing; similar to randomly connecting cables between your TV and DVD player. This is a common strategy no doubt which, more often than not, fails to produce a picture on the screen. If a chip's behavior cannot be predicted and controlled, it cannot be programmed. [8] Michael Conrad has argued, in a paper heavily drawn upon by Kittler, that this situation creates a necessary trade-off between connectivity and programability. "The amount of information processing carried out by a physical system freed from the constraints necessary to support programmability is thus potentially much greater than the potential information processing performed by a system not so constrained." [9] Taking this to its extreme conclusion, Kittler argues that only by removing the restrictions necessitated by programability is it possible to "enter into that body of real numbers originally known as chaos." [10]

Kittler radicalizes Conrad's argument and describes non-programmable machines as "badly needed" in that they "work essentially on a material substrate whose connectivity would allow for cellular reconfigurations" and so, "Software in the usual sense of an ever-feasible abstraction would not exist any longer." Kittler's brash materialism again shines through. Only when procedures are moved in a non-symbolic way to the real of the material, when the matter of the chip always and only executes the same operation as a matter of material necessity, have we finally created an authentic computing machine.

Despite being non-programmable, the machines described by Conrad are still usable for human tasks. However, these machines would rely on evolutionary techniques to find solutions to a problem. The programmer then becomes a breeder, combining elements from the best individuals to create a new generation, designing environmental fitness conditions and running genetic operations in an iterative process of searching the terrain of possible solutions. Adrian Thompson created just this type of system by working with field-programmable gate arrays (FPGA) – a type of integrated circuit that is physically reconfigurable. He developed an evolutionary system to evolve a configuration of an FPGA chip capable of discriminating between two audible tones. Eventually a successful configuration emerged. But unlike a solution expected from a programmable system, the evolved configuration took advantage of unique material properties of the chip on which it evolved, using quantum tunneling and exploiting irregularities in the physical material of the chip. As Thompson puts it, "a robust asynchronous design was found that could not have resulted from normal design principles." [11]

An evolutionary system using non-programmable chips does have far fewer layers of abstraction and obfuscation between the programmer and the machine. But if, unlike Kittler, we resist the temptation to confuse matter with medium, then we are not compelled to interpret this as a necessarily more authentic engagement with computation. It is simply another way for humans to think through and use computational systems. Thompson's work, though unlike other acts of programming, nevertheless involved the articulation of a process–or perhaps a meta-process–in the form of an evolutionary system working to create a FPGA configuration capable of a specific task. In other words, the medium of the programmer is process in the form of code, not always (or only) the hardware on which her software is executed.

DOING / BEING

It is nearly impossible to talk about coding without talking about what the coder is trying to accomplish. The discussion is always already shot through with mentions of goals, tasks, problems, intentions, and action. Even the evolutionary techniques as applied to so-called non-programmable hardware require a specific formulation of human goals. We are still asking the machine to perform an operation, just in a different way and using a different vocabulary. Even in the most software-free variation, the trail of the human serpent runs over the voltages in the machine. In software there are always many moments of doing: The back-and-forth between machine and coder as software is written, the compiling of that source code into opcodes for the machine, the effect of running the code on the machine. Eventually all running software must rub up against the needs and goals of the user, though these needs may have been prefigured in advance by the assumptions of the coder. Of course many times this loop is closed as the programmer herself becomes a user of her own creations. To identify one moment in this chain as the essential moment – in Kittler's case, when the opcodes finally control voltages – is to attempt to replace doing with being.

ISSN 1071-4391 ISBN 978-1-906897-16-1
ISSN 1071-4391 ISBN 978-1-906897-16-1

And to do so in the most brutally materialist way; if software does not exist, there is nothing to be said about the effect software has on politics or thought, either the thought of the coder or of the end-user. A virus that destroys a nation's economy, a protein folding simulation that finds the cure for a disease, and a copy of Minesweeper all do the same thing; they create voltages in silicon chips. Had they not been written in code, we could not talk about them in terms of politics, social change, or ethical import. They would simply be or not. ■

REFERENCES AND NOTES

1. Though the vast majority of code is text-based, meaning it is written using the characters and symbols common to the ASCII specification, code can also be visual, in which the processes of "writing" entails connecting graphical elements called patches much like programming a modular analog synthesizer or connecting an electronic circuit. The most well know patch-based environment is MAX/MSP, but newer patching systems include VVVV, Quartz Composer, and Grasshopper. Given Kittler's revulsion at the graphical user interface ("[O]n an intentionally superficial level, perfect graphic user interfaces, since they dispense with writing itself, hide a whole machine from its users") it is fair to assume that visual programming languages would be met with considerable scorn.
2. Friedrich Kittler, "There is No Software," in *Literature, Media, Information Systems*, ed. John Johnston (New York: Routledge, 1997), 147–155.
3. These values aren't perfect; i actuality a voltage range is broken roughly into thirds with the upper and lower thirds representing 1 and 0 respectively and the middle third denoting an indeterminate state.
4. Florian Cramer takes issue with the use of the term "natural language" as opposed to "formal" or "programming" language. I take "natural" in this context to refer more to the development and etymology of spoken languages vis-à-vis programming languages rather than a claim about ontological status. Florian Cramer, "Language" in *Software Studies*, ed. Matthew Fuller (Cambridge: MIT Press, 2008), 168–174.
5. Michel Foucault,"Language to Infinity" in *Language, Counter-Memory, Practice*, ed. Donald Bouchard (Ithaca: Cornell University Press, 1980), 53–67.
6. Noah Wardrup-Fruin explores this idea and the expressive potential of software in his book Expressive Processing. Noah Wardrip-Fruin, *Expressive Processing* (Cambridge: The MIT Press, 2009).
7. Friedrich Kittler, "There is No Software," in *Literature, Media, Information Systems*, ed. John Johnston (New York: Routledge, 1997), 147–155.
8. More accurately this type of chip may be programmable, but to do so would require techniques specific to each individual chip.
9. Michael Conrad, "The Price of Programability" in *The Universal Turing Machine: A Half-Century Survey*, ed. Rolf Herken (New York: Springer-Verlag, 1995), 261–282.
10. Friedrich Kittler, "There is No Software," in *Literature, Media, Information Systems*, ed. John Johnston (New York: Routledge, 1997), 147–155.
11. Adrian Thompson,"Notes on Design Through Artificial Evolution: Opportunities and Algorithms" in *Adaptive Computing in Design and Manufacture*, ed. I. C. Parmee (New York: Springer-Verlag, 2002), 17–26.

 ISSN 1071-4391 ISBN 978-1-906897-16-1