EXE.CUT[UP]ABLE STATEMENTS: THE INSISTENCE OF CODE

FLORIAN CRAMER

THE RESURGENCE OF CODE

Friedrich Nietzsche's aphorism that "our writing tools are also involved in our thoughts", put down after he first used a mechanical typewriter, has become an axiom of contemporary media theory. Applied to computer code and art, it could also be phrased into a question: How does code shape the aesthetics of digital art and technology?

"10 Programs written in BASIC ©1984", the title of a solo show of the Dutch-Belgian artists Jodi in Malmö/Sweden, June 2003, can be read beyond its specific description of the work exhibited as a programmatic statement that leads the digital arts back to the future. Consisting of vintage 1980s Sinclair ZX Spectrum home computers running, quote, "ten programs in BASIC that the visitor can add and change things to", the exhibition shows jodi's work at a radical peak of a development that began with net art and continuously evolved from web browser manipulations through HTML data, manipulations of computer games in sourcecode to increasingly abstract "variations" of simple BASIC program code, dismantling the fact that, despite the boom of graphical user interfaces in the 1980s and 1990s, program code not only remains at the heart of computing and digital information technology, but also that its structure hasn't changed since 1984, the year the Apple Macintosh computer was first sold. While artists used to run software code in black boxes to drive work that didn't show the programming involved — interactive video installations, for example —, since the late 1990s, programming and program code have received increasing attention as artistic material in themselves.[1] Today, the Apple Macintosh has likewise mutated into a flavor of the Unix operating system which no longer hides its codes, but exposes it on a textual commandline interface, thus catching up with contemporary digital arts and its increasing use of computer code as material.

The most radical conception of computer code as artistic language might be found "codeworks",[2] Net art which typically circulates as E-Mail and is written in private languages that hybridize English, program and network code and visual typography, like in the following example:

```
Exe.cut[up]able statements ---
::do knot a p.arse.r .make.
::reti.cu[t]la[ss]te. yr. text.je[llied]wells .awe. .r[b]ust.
```

Date: May 30, 2003.

[1] As obvious not only in the example of Jodi, but more generally in the establishing of "software art" as a genre description on international festivals and exhibitions of contemporary electronic arts such as transmediale (Berlin), readme (Moscow/Helsinki), CODeDOC (Whitney Museum, New York).

[2] First systematically covered in the "Codework" special issue of the American Book Review, see [Son01]

In these lines, which were taken from the posting “Re: OPPO.S[able].I.T[humbs]ION!!” by the Australian net artist mez (Mary Anne Breeze) to the mailing list “arc.hive” on January 14th, 2003, the word „Exe.cut[up]able“ becomes a self-describing executable source-code which expands into, among others, “Exe” (the Microsoft DOS/Windows file extension for executable programs), “Executable” and, recalling the mechanized collage techniques of Brion Gysin and William S. Burroughs, “cutup”. A poetic employment of computer code which adheres more literally to Unix, Perl or C syntax can be found in E-Mail code-works of Alan Sondheim, jodi, Graham Harwood, Johan Meskens and Pascale Gustin.

Aside from the more narrow field of digital art and technology in which code appears as code, it can first be observed that the graphical interface paradigm in so-called end-user software scales only to a limited amount of complexity, thus creating the need for textual scripting extensions and programming interfaces in the software itself;[3] secondly, that artists (and also non-artists) who employ software for complex tasks and as part of their thinking and subjectivity tend to push its limits to the point where they start to extend, re-program or re-implement its code. This tendency manifests itself today in two phenomena, the one being Free (or Open Source) Software as it has gained the attention of a larger public since circa 1998, the other one being software art, a topic in the digital arts since circa 2000, with both discourses intersecting in such projects as runme.org, a software art website created after the model of Free Software download repositories, and sweetcode.org, a Free Software website referencing conceptually (and hence also artistically) interesting computer programs.

FROM UTOPIA TO END-USER COMPUTING

Investigating the insistence of code in computer-based art and technology is pointless, however, if one does not question what computer code exactly is beyond its conventional surface appearance as plain text instructions. Beyond that, program code can be put down in any notation, in flowcharts or graphical simulations of circuitery for example,[4] or simply as a chain of zeros and ones. The problem of its representation does not only concern media in particular, but linguistics and semiotics in general: How can language in its complexity, cultural particularity and oddity be boiled down to an easy-to-grasp international system?

The question is much older than computer interface research, having been addressed in, among others, the Renaissance political and educational utopias of Tommaso Campanella’s “City of the Sun” and Jan Amos Comenius’ “Orbis pictus”. In the ideal city imagined by Campanella, all knowledge is being conveyed to the public via a graphical user interface:

> It is Wisdom who causes the exterior and interior, the higher and lower walls of the city to be adorned with the finest pictures, and to have all the sciences painted upon them in an admirable manner. [...] There are magistrates who announce the meaning of the pictures, and boys are accustomed to learn all the sciences, without toil and as if for pleasure; but in the way of history only until they are ten years old.

5

[3] “Scripting” being nothing but another word for programming.

[4] Such as in the musicial composition frameworks “MAX’ and “PD” designed by Miller Puckette

[5] Tommaso Campanella, City of the Sun [Cam82]

The hermetic philosopher and famous educational reformer Comenius translated this idea into the "Orbis pictus", the first illustrated children's book and canonical school text in 17th and 18th century Europe, which taught the whole inventory of macro- and microcosm simultaneously through pictures, in Latin words and in the pupils' native language. However, the design flaws of these pictorial languages, and the dilemma of linguistic operations through graphical user interfaces were addressed already in 1706 in the "Grand Academy of Lagado" chapter of Jonathan Swift's "Gulliver's Travels":

> The other project was a scheme for entirely abolishing all words whatsoever [...]. An expedient was therefore offered, that since words are only names for things, it would be more convenient for all men to carry about them such things as were necessary to express the particular business they are to discourse on. [...] However, many of the most learned and wise adhere to the new scheme of expressing themselves by things, which has only this inconvenience attending it, that if a man's business be very great, and of various kinds, he must be obliged in proportion to carry a greater bundle of things upon his back, unless he can afford one or two strong servants to attend him. I have often beheld two of those sages almost sinking under the weight of their packs, like pedlars among us; who, when they met in the streets, would lay down their loads, open their sacks, and hold conversation for an hour together; then put up their implements, help each other to resume their burdens, and take their leave.[6]

In other words: A new language in which object and sign are similar or identical is believed to be superior to traditional symbolic language with its abstract object-sign relationships. While the paragraph reads like a straightforward critique of iconic graphical computer interfaces, this critique does not actually concern images as opposed to text, but ungrammatical versus grammatical language as means of communication. The objects which the speakers employ cannot be linked or condensed in a meaningful way to express more complex semantic operations. In the realm of computer operating systems, this translates into an opposition less of graphical versus textual, but of programmer-friendly versus programmer-hostile environments.

When Alan Kay developed the first computer controlled by screen icons and the mouse at Xerox PARC labs in the 1970s, the separation between "usage" and "programming" was for the first time implemented as separation of media: "Usage" became graphical, "programming" textual. The gap widened with the commercialization of Kay's ideas through the Apple Macintosh and Microsoft Windows. While Alan Kay's user interface was based on the programming language Smalltalk and remained fully programmable to the point where users could create their own applications by combining pre-existing and self-written program objects,[7], the Apple Macintosh lacked the programming interface simply for economical and marketing reasons. Without Smalltalk, the system could run fast on low-power hardware, thus dropping sales prices and decreasing development costs. The result was a Swiftian ungrammatical operating system that gave birth to the "user", with the message:

[6] [Swi60], p. 183

[7] A concept living on — and currently being revived chiefly by artists working with the system — in "Squeak", the latest incarnation of Kay's Smalltalk system, released a cross-platform Free/Open Source software

You are supposed to read, not to write.[8] Only through this engineered gap, programming becomes a mystery, a black art, supposedly for the sake of making computing easier. Borrowing from Roland Barthes' book "S/Z" from 1970, GUI operating systems could be called "readerly" and command line-centric operating systems like Unix "writerly". According to Barthes, the readerly text presents itself as linear and smoothly composed, "like a cupboard where meanings are shelved, stacked, safeguarded".[9] Reflecting in contrast the "plurality of entrances, the opening of networks, the infinity of languages",[10] the writerly text aims to make "make the reader no longer a consumer, but a producer of the text".[11]

WRITERLY COMPUTING

Barthes' distinction shows why projecting the question how "our writing tools are also involved in our thoughts" onto software is problematic in itself: precisely because it projects issues of pre-digital hardware onto digital software. Nietzsche's observation of the mechanical typewriter naturally assumes a clear-cut division, a material difference between the tool and the writing, the processor and the processed; a division however which no longer exists in software since computers adopted the von Neumann architecture of storing both instruction code and data in the same physical and symbolic realm. Since computer software is tools made from writing, processors made from code, the old material gap can only be sustained through simulation. To be readerly, popular PC user software creates the illusion of being hardware, visually and tactically diguising itself as solid analog tools.[12] Graphic design software like Adobe Photoshop or Illustrator have palettes and pencils (which are subverted in Adrian Ward's software artworks "Autoshop" and "Auto-Illustrator"), web browsers pretend to be nautical navigation instruments ("Navigator" and "Explorer" by Netscape and Microsoft), word processors like Microsoft Word are based on visual simulations of a piece of paper in a typewriter.[13]

The attributes of the "writerly" on the other hand exist on the Unix commandline as a hybridity of codes. Since almost everything in Unix is ASCII text — from the user programs which are executed as word commands and its feedback messages to the data flowing between the commands — anything can be instantly turned into anything, writing into commands and command output into writing:

```
CORE CORE bash bash CORE bash


There are %d possibilities.  Do you really
wish to see them all? (y or n)

SECONDS
SECONDS
```

[8] Which, by the way, doesn not apply only to propietary, but also to Free/Open Source graphical user interfaces like KDE and Gnome.

[9] [**?**], p. 200

[10] [**?**], p. 5

[11] [**?**], p. 4

[12] Alan Kay accordingly states in a 1990 paper that "at PARC we coined the phrase *user illusion* to describe what we were about when designing user interface" [Kay's emphasis], [Kay90], p. 199

[13] See also Matthew Fuller's cultural analysis of Microsoft Word in [Ful01]

```
grep hurt mm grep terr mm grep these mm grep eyes grep eyes mm grep hands
mm grep terr mm > zz grep hurt mm >> zz grep nobody mm >> zz grep
important mm >> zz grep terror mm > z grep hurt mm >> zz grep these mm >>
zz grep sexy mm >> zz grep eyes mm >> zz grep terror mm > zz grep hurt mm
>> zz grep these mm >> zz grep sexy mm >> zz grep eyes mm >> zz grep sexy
mm >> zz grep hurt mm >> zz grep eyes mm grep hurt mm grep hands mm grep
terr mm > zz grep these mm >> zz grep nobody mm >> zz prof!

if [ "x`tput kbs`" != "x" ]; then # We can't do this with "dumb" terminal
        stty erase `tput kbs`

DYNAMIC LINKER BUG!!!
```

The above is an experimental codework by Alan Sondheim, generated from interaction with a Unix-compatible text environment running “bash”, the standard commandline interpreter of GNU/Linux operating systems. The medium of plain text allows to migrate all writing transparently from one symbolic form — the operating system — to a second one — E-Mail — all the while involving the third symbolic form of electronic word processing. The codes it contains are at English writing, system instruction code and Internet communication code simultaneously.

This conflation of codes is not a particular artistic invention of Sondheim, but a standard feature of the Unix operating system. The instantaneous mutual convertibility of processed data into algorithmic processors that is characteristic of all program code is not suppressed on the Unix commandline by hiding the code away, but is preserved on the level of the user interface. It is, as a matter of act, a system feature which Unix administrators rely on every day. Unix, like other operating systems which use written code not just as a hidden layer driving things underneath, but also as their user interface,[14] is a giant, modular and recursive text processing system which runs on scripts and parameters filtered through themselves; or, as the programmer and system administrator Thomas Scoville puts it in his 1998 paper “The Elements Of Style: UNIX As Literature”: “UNIX system utilities are a sort of Lego construction set for word-smiths. Pipes and filters connect one utility to the next, text flows invisibly between. Working with a shell, awk/lex derivatives, or the utility set is literally a word dance.”[15]

While it is true for all software and all operating systems that the software tool itself is nothing but writing, the elegant simplicity of the Unix commandline relies on the idea that programming, instead of becoming a secluded application, is a trivial extension of “using” the system, simply by writing a sequence of commands into a text file which then can be executed. Operating systems working in other than textual media in contrast haven’t found a way yet to make data and processing interchangeable and extend graphical user interfaces to graphical programming interfaces. Even Alan Kay concedes that “it would not be surprising if the visual system were less able in this area [of programmation] than the mechanism that solve noun phrases for natural language. Although it is not fair to say that ‘iconic languages can’t work’ just because no one has been able to design a good one, it is

[14] Other sophisticated examples being ITS, LISP machines, Plan 9, with DOS and 8-bit BASIC computers being comparatively primitive implementations of the concept

[15] Thomas Scoville, The Elements of Style: Unix As Literature, [Sco98]

likely that the above explanation is close to truth" — a truth explaining why alphanumeric text remains the most elegant notation system for computer instructions.[16]

IMAGINARY COMPUTING

If a medium is defined as something in between a sender and a receiver, i.e. a transmission channel or storage device, then computers are not just media, but also senders and receivers which themselves are capable of writing and reading, generating, filtering and interpreting messages within the limitations of the formal rule sets inscribed into them.[17] Alan Kay's outcry "the computer is a medium!"[18] explains why his system fails to provide more than a rudimentary grammar and a fixed set of abstractions — as opposed to the Turing-complete grammar and programmable abstractions of, for example, Unix shells —, encompassing only prefabricated algorithmic processors ("user programs") and storage and transmission operations like "open", "save", "close", "send", "cut", "copy", "paste".

Not surprisingly, art forms which reflect the computer beyond its functioning as a medium were coined as artisanship by programmers in textual notation: programming language poetry (like Perl poetry), code slang (generated by manipulating text with algorithmic filters), viral scripting, in-code recursions and ironies (as for example in the self-replicating sourcecode of "Quines"), all of them being largely independent of particular transmission and storage media or output technology – which include computer screens, voice output, print books and t-shirts alike –, and all of them having been systematically appropriated in the experimental digital arts since the late 1990s. The hacker "Jargon File"[19], whose first versions appeared around 1981 at the MIT, is therefore to be read not only as the poetics of these forms, but also as a blueprint for net cultures and net arts since the mid-1990s. When, as Thomas Scoville writes, instruction vocabulary and syntax like that of Unix becomes "second nature",[20] it follows that formal turns into conversational language, and syntax into semantics. Beyond being a "tool" merely "involved" in shaping thoughts, computer language becomes a way of thinking and mode of subjectivity, making even such code exe.cut[up]able which cannot run but in the reader's imagination.[21]

(Thanks to Saul Albert, Josephine Bosma, Robbin Murphy and Frederic Madre for comments and corrections)

REFERENCES

[Cam82] Tommaso Campanella. *The City of the Sun*. Harpercollins, 1982.

[Ful01] Matthew Fuller. It Looks Like You're Writing a Letter: Microsoft Word, 2001. `http://www.heise.de/tp/english/inhalt/co/7073/1.html`.

[16][Kay90], p. 203

[17]Unless one would take a radical humanist view of "medium" in the sense of machine processes being nothing but mediations between human individuals.

[18]A "shock", he writes, caused by reading McLuhan, [Kay90], p. 193

[19]Currently maintained by Eric S. Raymond, [rayoJ]

[20][Sco98], ibid.

[21]Jonathan Swift anticipates this as well when Lemuel Gulliver says about the academics of the flying island of Lagado: "The knowledge I had in mathematics gave me great assistance in acquiring their phraseology, which depended much upon that science and music; and in the latter I was not unskilled. Their ideas are perpetually conversant in lines and figures. If they would, for example, praise the beauty of a woman, or any other animal, they describe it by rhombs, circles, parallelograms, ellipses, and other geometrical terms, or by words of art drawn from music, needless here to repeat", [Swi60], p. 157

[Kay90] Alan Kay. User Interface: A Personal View. In Brenda Laurel, editor, *The Art of Human-Computer Interface Design*, pages 191–207. Addison Wesley, Reading, Massachusetts, 1990.

[rayoJ] The Jargon File, o.J. http://www.catb.org/jargon/.

[Sco98] Thomas Scoville. The Elements of Style: Unix as Literature, 1998. http://web.meganet.net/yeti/PCarticle.html.

[Son01] Alan Sondheim. Introduction: Codework. *American Book Review*, 22(6), September 2001.

[Swi60] Jonathan Swift. *Gulliver's Travels*. Washington Square Press, New York, 1960.