Exe.cut[up]able statements: Das Drängen des Codes an die Nutzeroberflächen

Florian Cramer

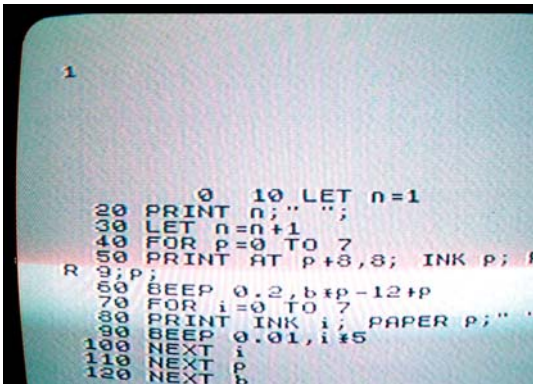
Auferstanden aus der Black Box

Auf Computer-Instruktionscodes projiziert, ließe sich Nietzsches zum medientheoretischen Axiom geronnene Erkenntnis aus seiner Begegnung mit der mechanischen Schreibmaschine, dass nämlich das „Schreibzeug mit an unseren Gedanken" arbeite, in eine Frage umformulieren: Wie prägt Programmcode Konzepte und Ästhetik digitaler Kunst und Technologie?

„10 Programs written in BASIC ©1984", dieser Titel einer Malmöer Ausstellung des holländisch-belgischen Künstlerduos Jodi vom Juni 2003 liest sich, über die technische Beschreibung der gezeigten Arbeit hinaus, als eine programmatische Aussage, die den digitalen Künsten den Weg zurück in die Zukunft weist. Indem sie nichts als eine Reihe zwanzig Jahre alter Heimcomputer des Typs Sinclair ZX Spectrum zeigt, auf denen „zehn BASIC-Programme [laufen], die der Besucher ergänzen oder verändern kann", markiert Jodis neueste Arbeit den radikalen Höhepunkt einer Werkgeschichte, die Mitte der 1990er Jahre mit Netzkunst beginnt, sich fortsetzt mit Web-Browser-Manipulationen durch trickreich codierte HTML-Daten, Quelltext-Manipulationen von Computerspielen bis hin zu immer abstrakteren „Variationen" kurzer Programme in der Programmiersprache BASIC. So zeigt Jodis Kunst nicht nur, dass ungeachtet der Karriere grafischer Nutzerführungen in den 1980er und 1990er Jahren Programmcode weiterhin die Basis aller Computeroperationen und digitaler Informationstechnologie bildet, sondern auch, dass seit 1984, jenem Jahr, in dem auch der Apple Macintosh auf den Markt kam, dieser Code ungeachtet aller äußerlichen Wandlungen der Software sich in Struktur und Notation sich kaum geändert hat. Während Künstler Softwarecode traditionell für schwarze Kästen schreiben ließen, die im Hintergrund versteckt Arbeiten steuerten, in denen von der Programmierung nichts mehr zu sehen war – wie z. B. im Genre der sogenannten interaktiven Video-Installationen –, wurde seit den späten 1990er Jahren Programmierung und Programmcode zunehmend selbst als künstlerisches Material begriffen und jene verlorengegangene Verbindung von Kunst und Software neugeknüpft, die 1970 Jack Burnhams New Yorker Konzeptkunst-Ausstellung „Software" herzustellen versuchte.[1] Historisch parallel zu dieser neueren Tendenz in den digitalen Künsten ist selbst der Apple Macintosh zu einer Spielart des Betriebssystems Unix mutiert, indem er seine Instruktionscodes nicht länger verbirgt, sondern auf einer Text-Kommandozeile exponiert, holt er gewissermaßem den status quo digitaler Kunst ein und ihres zunehmenden Gebrauchs von Computercode als Material.

Das radikalste Verständnis von Computercode als künstlerischem Material zeigt sich in „Codeworks",[2] einer Netzkunst, die zumeist als E-Mail zirkuliert und in Privatsprachen geschrieben ist, die, wie im folgenden Beispiel, Englisch, Programm- und Netzwerkcodes mit visueller Typografie vermischt:

```
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.
```



Jodi, Screenshot der Installation *10 Programs written in BASIC ©1984,* Malmö 2003

Die Zeilen stammen aus einer Arbeit der australischen Netzkünstlerin mez (Mary Anne Breeze), die am 14. 1. 2003 als E-Mail über die Mailingliste „arc.hive" verschickt wurde. Das Wort „Exe.cut[up]able" wird in ihr zu einem Quellcode, der sich in seiner Ausführung selbst beschreibt: „Exe", die Microsoft DOS/Windows-Dateikennung ausführbarer Programme, „Executable" und, ein Verweis auf die automatisierten Montagetechniken von Brion Gysin und William S. Burroughs, „cutup" sind unter den Wörtern, in die dieses Notat expandiert. Die E-Mail-Codeworks von zum Beispiel Alan Sondheim, jodi, Graham Harwood, Johan Meskens und Pascale Gustin halten sich in ihrer poetischen Aneignung von Programmcode noch genauer an die Syntax realer Programmiersprachen wie der Unix-Shell, Perl und C.

Auch abseits des Spezialgebiets digitaler Kunst und Technologie, die Quellcode als Quellcode erscheinen lässt, ist erstens zu beobachten, dass das Paradigma der grafischen Bedienung sogenannter Nutzersoftware nur bis zu einem begrenzten Grad von Komplexität skaliert und deshalb in der Software selbst Skripting-Erweiterungen und Programmierschnittstellen nötig macht, die sich textuell präsentieren.[3] Zweitens fällt auf, dass Künstler (ebenso wie Nicht-Künstler), die Computersoftware für komplexe, individualisierte Anwendungen einsetzen und als Teil ihres Denkens und ihrer Subjektivität verstehen, regelmäßig an Grenzen fertiger Programme stoßen und deshalb beginnen, ihren Code zu erweitern, umzuprogrammieren oder neuzuschreiben, eine Tendenz, die sich in zwei Phänomenen manifestiert: der Freien (bzw. Open Source) Software, wie sie seit circa 1998 größere öffentliche Aufmerksamkeit erfahren hat, und der Softwarekunst, die ungefähr seit dem Jahr 2000 als Genre digitaler Kunst diskutiert wird. Beide Diskurse überschneiden sich zum Beispiel in Projekten wie *runme.org,* einer Softwarekunst-Website, die auf der technischen Grundlage Freier Software nach dem Vorbild von Download-Servern Freier Software gestaltet ist und auch viele Freie-Software-Projekte verzeichnet, sowie *sweetcode.org,* einer Freien-Software-Website, die konzeptuell ungewöhnliche – und daher oft auch künstlerisch interessante – Computerprogramme verzeichnet.

Von Utopia zum User Computing

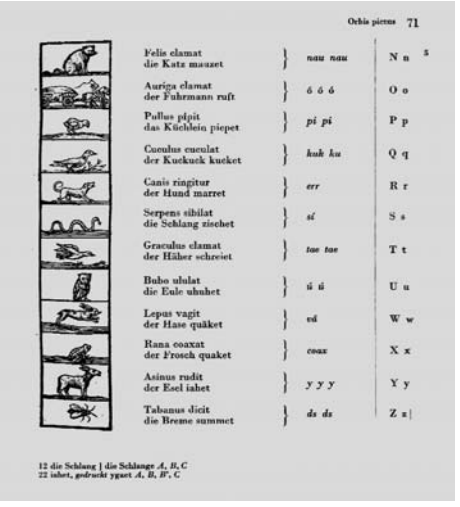
Das Drängen des Codes an die Oberflächen computergestützer Kunst und Technologie zu untersuchen wäre jedoch sinnlos, ohne zu fragen, was genau Computercode ist jenseits seiner konventionellen Erscheinung als in ASCII-Text notierten Instruktionen. Denn schließlich kann er auf jede denkbare Weise notiert werden, z. B. auch in Flussdiagrammen oder als grafische Simulation von Schaltungen[4] oder einfach als Kette von Nullen und Einsen. Dieses Darstellungsproblem ist jedoch nicht nur ein medientheoretisches, sondern allgemein linguistisches und semiotisches: Wie kann eine Sprache in ihrer Komplexität und kulturellen Besonderheit zu einem international allgemeinverständlichen System vereinfacht werden?

Prominent wurde diese Frage bereits in frühneuzeitlichen Politik- und Bildungsutopien wie Tommaso Campanellas *Sonnenstaat,* dessen visionierten Interface Jan Amos Comenius praktisch implementierte. Den Einwohnern von Campanellas idealen Stadtstaat wird alles Wissen über eine grafische Nutzeroberfläche bereitgestellt:

> Der „Weisheit“ hat die Mauern der ganzen Stadt von innen und außen, unten und oben mit herrlichen Gemälden schmücken und auf ihnen so alle Wissenschaften in fabelhafter Anordnung wiedergeben lassen. […] Sie haben Lehrer, die all dies Bilder erklären, und die Kinder pflegen noch vor dem zehnten Lebensjahre ohne große Mühe, gleichsam spielend […] alle Wissenschaften zu lernen.[5]

Der Erziehungsreformer und hermetische Philosoph Comenius übersetzte dieses Konzept in seinen *Orbis pictus,* der nicht nur das erste illustrierte Kinderbuch ist, sondern bis ins späte 18. Jahrhundert die kanonische Schulfibel Europas. Alle Dinge des Makro- und Mikrokosmos inventarisiert der *Orbis pictus,* indem er sie simultan auf Holzschnitten und als Wörter auf Latein und in der jeweiligen Muttersprache der Schüler verzeichnet. Die Designfehler dieser Bildsprachen und das Dilemma linguistischer Operationen durch grafische Nutzerinterfaces wurden früh bemerkt. 1706 heißt es in Jonathan Swifts satirischen *Gulliver's Travels* über die Forschungen an der Akademie von Lagado:

> Das zweite Projekt war ein Plan zur völligen Abschaffung aller Wörter überhaupt, […] da Wörter nur Bezeichnungen für Dinge sind, sei es zweckdienlicher, wenn alle Menschen die Dinge bei sich führten, die zur Beschreibung der besonderen Angelegenheit, über die sie sich unterhalten wollen, notwendig seien. […] Viele der Gelehrtesten und Weisesten sind jedoch Anhänger des neuen Projekts, sich mittels Dingen zu äußern; das bringt nur die eine Unbequemlichkeit mit sich, dass jemand, dessen Angelegenheiten sehr umfangreich und von verschiedener Art sind, ein entsprechend größeres Bündel von Dingen auf dem Rücken tragen muß, falls er es sich nicht leisten kann, dass ein oder zwei starke Diener ihn begleiten. Ich habe oft gesehen, wie zwei dieser Weisen unter der Last ihrer Bündel fast zusammenbrachen, wie bei uns die Hausierer. Wenn sie sich auf der Straße begegneten, legten sie ihre Lasten nieder, öffneten ihre Säcke und unterhielten sich eine Stunde lang; dann packten sie ihre Utensilien wieder ein, halfen einander, ihre Bürden wieder auf den Rücken zu nehmen, und verabschiedeten sich.[6]



Jan Amos Comenius (Komensky), Seite aus dem *Orbis pictus*

Was sich wie eine pointierte Kritik ikonischer Computer-Nutzerinterfaces liest, betrifft jedoch weniger das Bild im Gegensatz zu Text, als ungrammatische versus grammatische Sprachen. Die Gegenstände, die hier an die Stelle der Wörter treten, können nicht sinnvoll verknüpft oder verdichtet werden, und sind somit auch semantisch nicht expressiv. Auf Computerbetriebssysteme übertragen, folgt daraus ein Gegensatz nicht von grafischen und textuellen, sondern vielmehr von programmierfreundlichen und programmierfeindlichen Softwareumgebungen.

Als Alan Kay in den 1970er Jahren im Xerox PARC-Labor den ersten per mausklickbare Bildschirm-Piktogramme steuerbaren Computer erfand, wurde der Unterschied von „Nutzung“ und „Programmierung“ von Computern erstmals auf medialer Ebene implemen-

tiert: die „Nutzung“ war nun grafisch, die „Programmierung“ jedoch blieb schriftsprachlich. Diese Kluft wuchs noch mit der Kommerzialisierung von Kays Konzepten durch den Apple Macintosh und Microsoft Windows. Während Kays Erfindung auf der Programmiersprache Smalltalk basierte und so programmierbar sein solte, dass Nutzer sich durch baukastenartige Kombinationen vorgefertigter und selbst geschriebener Programmodule (bzw. -objekte) ihre Anwendungen selbst schaffen konnten,[7] fehlte dem Apple Macintosh diese Programmierschnittstelle aus simplen ökonomischen Gründen: Ohne Smalltalk konnte das System ausreichend schnell auf langsamer Hardware laufen, was sowohl Verkaufspreise, als auch Entwicklungskosten reduzierte. Das Ergebnis war ein ungrammatisches Betriebssystem, das den „User“ gebar und ihm bedeutete, dass er das System zu lesen und nicht zu schreiben habe.[8] Nur durch die konstruierte mediale Kluft wurde das Programmieren zum Mysterium, zur schwarzen Kunst. Mit einem Begriffspaar, das Roland Barthes in seinem Buch *S/Z* von 1970 einführt, könnte man Betriebssysteme mit grafischer Oberfläche „leserlich“ („lisible“) nennen, Kommandozeilen-zentrische wie Unix hingegen „schreiberlich“ („scriptible“). Für Barthes präsentiert sich der „leserliche“ Text als linear und bruchlos gefügt, „wie ein Haushaltsschrank, in dem die Sinngebungen geordnet, aufeinandergestapelt, gehortet sind“,[9] während der antiklassische „schreiberliche“ Text, indem er die „Offenheit des Textgewebes, die Unendlichkeit der Sprachen“[10] reflektiert, aus dem „Leser nicht mehr einen Konsumenten, sondern einen Textproduzenten“ macht.[11]

Writerly Computing

An Barthes’ Begriffspaar zeigt sich auch, dass die Frage, wie unser Schreibzeug an unseren Gedanken mitarbeitet, problematisch wird, sobald man sie auf Computersoftware anwendet, eben weil sie Eigenschaften prädigitaler Hardware auf Software projiziert. So liegt auch Nietzsches Beobachtung der mechanischen Schreibmaschine noch die Annahme zugrunde, dass Werkzeug und Schrift, Prozessor und Prozessiertes materiell voneinander verschieden sind, eine Trennung, die in der Computersoftware nicht mehr gilt, seit Computer mit der von Neumann-Architektur sowohl Instruktionscode, als auch Daten im selben physikalischen und symbolischen System speichern. Da Computersoftware aus Schrift gefertigtes Werkzeug ist, ein Prozessor aus Instruktionscode, lässt sich die alte Grenze des Materials nur durch Simulation aufrecht erhalten. Um im Sinne Barthes’ leserlich zu sein, tut PC-Software so, als wäre sie Hardware, indem sie sich als visuell und taktil solides Werkzeug darstellt.[12] Grafikdesign-Software wie Adobe Photoshop oder Adobe Illustrator wird mit simulierten Pinseln und Paletten bedient (deren vermeintliche Selbstverständlichkeit Adrian Wards Softwarekunstwerke *Autoshop* und *Auto-Illustrator* gezielt subvertieren), Web-Browser wie der Navigator von Netscape und der Explorer von Microsoft geben vor, nautische Instrumente zu sein, und Textverarbeitungsprogramme wie Microsoft Word basieren auf der optischen Simulation von Papier in einer Schreibmaschine.[13]
Die Eigenschaften des „schreiberlichen“ Textes hingegen materialisieren sich auf der Unix-Kommandozeile als Hybridität der Codes. Da in Unix fast alles ASCII-Text ist – von den als Kommandowörtern ausgeführten Programmen und ihren Rückmeldungen bis zu den Daten, die zwischen ihnen fließen –, kann alles augenblicklich in alles beliebige umgewandelt werden, Schrift in Kommandos und Kommando-Ausgaben in Schrift:

```
CORE CORE bash bash CORE bash

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

SECONDS
SECONDS
```

```
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!!!
```

Diese Zeilen, die den Anfang eines „Codework" von Alan Sondheim bilden, protokollieren eine Interaktion mit einer Unix-kompatiblen Textumgebung, deren Motor hier die „bash", der Standard-Befehlsinterpreter des GNU / Linux-Betriebssystems bildet. Das Einheitsmedium des ASCII-Texts erlaubt, ein und denselben Code transparent von einer symbolischen Form (dem Betriebssystem) in eine andere (E-Mail) zu schreiben, und dies mit Hilfe der ins System integrierten algorithmischen Textverarbeitung als dritter symbolischer Form. So ist der Code zugleich englischer Text, Systemkommando und Kommunikationscode im Netz.
Diese Vermischung der Codes ist nicht Sondheims künstlerische Erfindung, sondern eine Standardfunktion des Unix-Betriebssystems. Die augenblickliche, reziproke Konvertierbarkeit von prozessierten Daten in algorithmische Prozessoren, die rekursive Prozessierung von Programmen durch sich selbst also, die ein Merkmal aller Programmiersprachen ist, wird auf der Unix-Kommandozeile nicht zum Verschwinden gebracht, sondern noch auf der Ebene des Bedieninterfaces beibehalten. Es ist sogar eine Systemfunktion, auf die Unix-Administratoren täglich zurückgreifen. Wie wenige andere Betriebssysteme, die schriftliche Instruktionen nicht nur als versteckte Schicht unterhalb der Software-Anwendungen, sondern auch als Bedieninterface implementieren,[14] ist Unix ein riesiges modulares und rekursives Textprozessierungssystem, das mit Skripten und Parametern läuft, die durch sich selbst gefiltert werden; oder, wie der Programmierer und Systemadministrator Thomas Scoville 1998 in seinem Aufsatz *The Elements Of Style: UNIX As Literature* schreibt: „Die Hilfsprogramme die Unix-Systems sind eine Art Lego-Baukasten für Wortschmiede. Pipes und Filter verbinden ein Utility mit dem anderen, Text fließt unsichtbar zwischen ihnen. Die Arbeit mit der Shell, awk/lex-Derivaten oder dem Werkzeugkasten der Hilfsprogramme ist buchstäblich ein Tanz mit Wörtern."[15]
Während es für alle Programme und alle Betriebssysteme zutrifft, dass die Software selbst nichts als eine Schrift ist, beruht die simple Eleganz der Unix-Kommandozeile auf der Idee, dass Programmierung keine vom System abgekapselte Anwendung ist, sondern eine triviale Erweiterung seiner alltäglichen „Nutzung", zum Beispiel indem man eine Abfolge von Kommandos in eine Textdatei schreibt, die dadurch als Programm ausführbar wird. Für Betriebssysteme hingegen, deren Bedienschnittstellen in anderen Medien als der Schrift implementiert sind, ist noch kein Weg gefunden worden, Daten und ihre Prozessierung austauschbar zu machen und grafische Nutzerinterfaces zu ebenfalls grafischen Programmierinterfaces zu erweitern. Sogar Alan Kay räumt ein, dass „es nicht überraschend wäre, wenn das visuelle System auf diesem Gebiet [der Programmierung] weniger leistungsfähig ist als der Mechanismus, der Wortphrasen verarbeitet. Obwohl die Behauptung nicht gerecht ist, dass ‚ikonische Programmiersprachen nicht funktionieren können' nur weil niemand bisher fähig war, eine gute zu konzipieren, ist es wahrscheinlich, dass die obengenannte Erklärung der Wahrheit nahekommt"; eine Wahrheit, aus der sich auch erklärt, weshalb alphanumerischer Text das eleganteste Notationssystem für Computerinstruktionen bleibt.[16]

Imaginary Computing

Definiert man als Medium etwas, das zwischen einem Sender und Empfänger steht, also ein Übertragungskanal oder Speicher ist, so sind Computer nicht nur Medien, sondern auch Sender und Empfänger, die Nachrichten selbst innerhalb der Grenzen ihrer eingeschriebenen formalen Regelwerke selbst schreiben und lesen, generieren, filtern und interpretieren können.[17] Aus Alan Kays Ausruf „the computer is a medium"[18] erklärt sich, weshalb seine Erfindung im Medium der ikonischen Bildschirmgrafik nicht mehr als eine rudimentäre Grammatik und als ein festgeschriebenes Set von Abstraktionen bereitstellt – im Gegensatz eben zur Turing-vollständigen Grammatik und den programmierbaren Abstraktion von Unix-Shells –, nur vorgefertige Anwendungen bietet und Speicher- und Übertragungsoperationen auf ein kleines Vokabular begrenzt: „open", „save", „close", „send", „cut", „copy", „paste".
So überrascht es nicht, dass zur Avantgarde einer digitalen Kunst, die Computer über ihre Funktion als Medien hinaus reflektiert, Vokabularien gehören, die als Programmierer-Folklore auf Kommandozeilen entstanden: Programmiersprachen-Lyrik (wie die *Perl poetry), Code-Slang* (durch algorithmisches Filtern von Text), virales Skripting, rekursiver und ironischer Programmcode (wie zum Beispiel im sich selbst exakt replizierenden Quellcode sogenannter „Quines"). Alle diese Formen sind weder auf spezifische Übertragungs- oder Speichermedien angewiesen, noch auf partikuläre Anzeigetechniken, sondern können sich gleichermaßen auf Bildschirmen, per Sprachsynthese und auf Büchern und T-Shirts materialisieren. Seit den späten 1990er Jahren wird sie systematisch in den experimentellen Digitalkünsten appropriiert. Das Hacker-„Jargon File",[19] dessen erste Versionen ca. 1981 am MIT entstanden, ist somit nicht nur als Poetik dieser Formen zu lesen, sondern auch als Blaupause von Netzkulturen und -künsten seit Mitte der 1990er Jahre. Wenn, wie Thomas Scoville schreibt, Unix-Vokabular und -Syntax Unix zur „zweiten Natur" werden kann,[20] folgt daraus, dass formale Sprachen zu Umgangssprachen werden, und Syntax zu Semantik. Über ein „Schreibzeug" hinaus, dass an Gedanken nur mitarbeitet, wird Computersprache zu einer Denkweise und zum subjektiven Register, in dem selbst solcher Code ausführbar – exe.cut[up]able – wird, der nur in der Imagination des Lesers abläuft.[21]

(Danke an Saul Albert, Josephine Bosma, Robbin Murphy und Frederic Madre für Anmerkungen und Korrekturen)

Referenzen siehe Seite 103