# DER SELBSTAUSFÜHRENDE ENTWURF: SOFTWARE UND SOFTWAREKUNST

FLORIAN CRAMER

## SOFTWARE UND KUNST

Seit den 1990er Jahren haben die Kunstwissenschaften Computer hauptsächlich als audiovisuelle Speicher-, Übertragungs- und Anzeigemedien betrachtet,[1] Reflexionen der der Schicht von Software-Steuercodes hingegen wenig Aufmerksamkeit geschenkt.[2] Analog wird digitale Kunst zumeist als Ablauf digitalisierter Bildern und Tönen, fast nie jedoch von Computerprogrammen beschrieben. „Interaktive Videoinstallationen“ zum Beispiel werden als „multimediales“ Ineinandergreifen von Betrachter, Ausstellungraum und Bildprojektion begriffen,[3] selten als Systeme auf der Basis algorithmischer Steuercodes, zumal ihre Software typischerweise als black box im Hintergrund auf versteckten Computern läuft. Er recht gilt dies für Kunst, der man äußerlich nicht ansieht, daß ihr Herstellungsprozeß Computerprogrammierung überhaupt einschloß. John Cages Hörspiel „Roaratorio“ von 1981 zum Beispiel erklingt als eine Tonbandmontage eines Sprechtexts, der auf Joyces „Finnegans Wake“ basiert, von Umweltgeräuschen, die in verschiedenen Städten der Welt aufgenommen wurden, und von irischer Volksmusik. Wie viele der Kompositionen von John Cage, ist „Roaratorio“ auch algorithmische Kunst; sein Text wurde wurde nach formalen Raster, Mesosticha des Namens „James Joyce“, aus dem Roman extrahiert, die Montage der Tonaufnahmen im Pariser IRCAM-Studio richtete sich nach einer computergenerierten Zufallspartitur. Die Buch-und-CD-Box von „Roaratorio“ dokumentiert die Komposition zwar detailliert und enthält das Hörstück selbst, einen Nachdruck des Sprechtexts, eine Aufnahme seiner Lesung durch John Cage, ein sowohl in Ton, als auch in Schrift wiedergegebenes Interview mit dem Komponisten einschließlcih einer Liste der

*Date*: 24.1.2002.

[1] So z.B. in [Lan92], [SB99] und [Gra01]

[2] Ausnahmen sind die Arbeiten von Abraham Moles [Mol71] und Max Bense [Ben62], und, neueren Datums, jene von Wolfgang Hagen und Matthew Fuller Wolfgang Hagen [Hag97], Matthew Fuller [Ful01]

[3] Stilbildend für dieses Genre digitaler Kunst waren Jeffrey Shaws Installationen im ZKM Karlsruhe, dazu s.a. [Gra01].

Städte, in denen Umgebungsgeräusche aufgenommen wurden. Die computergenerierte Partitur aber ist nur in einem einseitigen Auszug faksimiliert, der Quellcode des Computerprogramms, das diese Partitur erzeugte, fehlt ganz.[4]

Die Geschichte der digitalen und computergestützten Künste handelt von Ignoranz gegenüber Software, von schwarze Kästen, und von als Künstler-Faktoten angestellten Programmiereren. Auch in den digitalen Künsten schreibt sich so klassisch-romantische Kunstideologie fort mitsamt ihrer Privilegierung der aisthesis gegenüber der poiesis.[5] Nicht minder problematisch ist die Selbstbezeichnung digitaler Künste als „(neue) Medienkünste“, der disparate Techniken wie Videobänder und Siliziumchips wahllos zusammenwirft. Wenn gealterte neue Medien wie Funk, Fernsehen, Video in Computer integriert werden, bleibt es dennoch ein zweifelhafter Umkehrschluß, den Computer dadurch insgesamt als „Medium“ zu identifizieren. Definiert man als Medium, was zwischen einem Sender und einem Empfänger steht, somit ein Übertragungskanal ist, dann sind Computer nicht bloß Medien, sondern selbst auch Sender und Empfänger, da sie in den Grenzen ihrer inskribierten formalen Regelwerke Information nicht nur übertragen, sondern auch schreiben und lesen, analysieren und filtern. Die Computersoftware zum Beispiel, die Kontostand und Überziehungskredit eines Girokontos berechnet oder medizinische Apparate in der Intensivstation eines Krankenhauses steuert, kann nicht mehr sinnvoll ein „Medium“ genannt werden, es sei denn, man deutet den Begriff radikal humanistisch und versteht alle Maschinenprozesse als mittelbare Kommunikationen menschlicher Individuen. Auch gibt es, strikt definitiert, keine digitalen Medien, sondern nur digitale Information. „Medial“ wird diese Information erst durch analoge Schnittstellen wie Bildschirme, Lautsprecher und Drucker, die Nullen und Einsen in analoge Schallwellen, Videosignale, Druckschwärze wandeln.

Gibt es somit auch eine poiesis digitaler Codes vor ihrer Wandlung in analoge Signale, eine poiesis, die nicht mehr sichtbar wäre, weil sie sich in den Chips von Rechenmaschinen abspielt? Sind diese Codes Entwürfe des späteren Outputs, oder gibt es auch Entwürfe von Codes?

---

[4] [Cag82] – Über computergenerierten Zufall sagt die Softwarekünstlerin Ulrike Gabriel, daß es ihn nicht gäbe, weil bereits die Maschine, die ihn berechnet, nicht zufällig da sei. Tatsächlich fehlt eine Kritik der Gleichsetzung von stochastischem und ontologischem Zufall in der Musik von John Cage und seinen Schülern.

[5] Siehe hierzu auch den Aufsatz „The Aesthetics of Generative Code“ von Geoff Cox, Adrian Ward und Alex McLean, [CWM01]

## WAS IST SOFTWARE?

Software besteht aus formalen Anweisungen, d.h. Algorithmen; sie ist eine formal-logische Partitur, notiert in einem codierten Zeichensystem. Ob dieser Code aus Nullen und Einsen, dem Dezimalsystem, dem römischen oder griechischen Alphabet, Morsecode, einem exakt definierten Repertoire von Stromspannungen oder den geschalteten Gattern eines Prozessorchips besteht, ist unerheblich, weil die Codierung austauschbar ist. Wenn Software also eine Partitur ist, ist sie dann per definitionem ein Entwurf im Sinne der Blaupause eines ausgeführten Werks?

Man stelle sich ein simples dadaistisches Gedicht auf der Grundlage von Hugo Balls „Karawane“ vor:

KARAWANE
jolifanto bambla ô falli bambla
grossiga m’pfa habla horem
égiga goramen
higo bloiko russula huju
hollaka hollala
anlogo bung
blago bung
blago bung
bosso fataka
ü üü ü
schampa wulla wussa ólobo
hej taat gôrem
eschige zunbada
wulebu ssubudu uluw ssubudu
tumba ba-umpf
kusagauma
ba-umpf

Das neue Gedicht könnte aus acht einfachen Variationen der Zeile „tumba ba-umpf“ bestehen. Der Aufführende würde für jede Zeile zweimal eine Münze werfen und, wenn sie auf die Kopfseite fällt, das Wort „tumba“ aufschreiben oder sprechen, und wenn die Zahl erscheint, das Wort „ba-umpf“. Ein mögliches Ergebnis wäre:

tumba tumba
ba-umpf tumba
tumba ba-umpf
tumba ba-umpf
ba-umpf ba-umpf

ba-umpf tumba
tumba ba-umpf
tumba ba-umpf

Eine genaue formale Anweisung zur Herstellung dieses Gedichtes könnte lauten:

(1) Nehmen Sie irgendeine Münze, deren zwei Seiten sich visuell klar voneinander unterscheiden.
(2) Legen Sie fest, welches für Sie die erste und welches die zweite Seite der Münze ist.
(3) Wiederholen Sie die folgenden Anweisungen achtmal:
  (a) Werfen Sie die Münze.
  (b) Fangen Sie die geworfene Münze so auf, daß sie auf einer ihrer beiden Seite landet.
  (c) Wenn die Münze auf die erste Seite fällt, führen Sie die folgende Anweisung aus:
    • Sagen Sie „tumba“
  (d) Anderenfalls führen Sie die folgende Anweisung aus:
    • Sagen Sie „ba-umpf“
  (e) Machen Sie eine kurze Pause, um das Ende der Zeile anzudeuten.
  (f) Werfen Sie die Münze.
  (g) Fangen Sie die geworfene Münze so auf, daß sie auf einer ihrer beiden Seite landet.
  (h) Wenn die Münze auf die erste Seite fällt, führen Sie die folgende Anweisung aus:
    • Sagen Sie „tumba“
  (i) Anderenfalls führen Sie die folgende Anweisung aus:
    • Sagen Sie „ba-umpf“
  (j) Machen Sie eine lange Pause, um das Ende des Gedichts anzudeuten.

Diese Instruktionen sind eindeutig genug, um auch von einer Maschine ausgeführt werden zu können, und deshalb Zeile für Zeile in ein Computerprogramm übersetzbar.In der Programmiersprache „Perl“, deren Grundsyntax leicht verständlich ist, könnte man die obigen Anweiseungen wie folgt schreiben:

```
for $line (1 .. 8) {
        $random_number = int(rand(2));
        if ($random_number == 0) {
                print "tumba"
```

```
                }
        else {
                print "ba-umpf"
                }
        print " "
        $random_number = int(rand(2));
        if ($random_number == 0) {
                print "tumba"
                }
        else {
i                print "ba-umpf"
                }
        print "\n"
        }
```

Der Münzwurf wird durch eine Zufahlszahl simuliert, die die Funktion *rand* („random“) ermittelt und die Funktion *int* („integer“) ganzzahlig abrundet. Weil *rand(2)* Zufallszahlen zwischen 0 und $1,\overline{9}$ ergibt, rundet *int* die Resultate auf Null oder eins.

```
print " "
```

und

```
print "\n"
```

geben ein Leerzeichen bzw. einen Zeilenumbruch aus. Das Programm kann auf fast jedem Computer ausgeführt werden; es ist eine einfache Software. Komplexere Programme wie zum Beispiele Betriebssysteme und Computerspiele unterscheiden sich nicht strukturell vom obigen Programmbeispiel, denn die Kontrollstrukturen – Variablen, Schleifen, logische Bedingungen – ähneln sich in allen Programmiersprachen. Der Unterschied liegt nur in der größeren Zahl und Verschachtelung der Instruktionen: Typische PC-Benutzersoftware kann umfaßt mehrere Millionen Zeilen Programmcode.

Anders als in der Anweisung, das Zufallsgedicht durch Münzwurf zu erstellen, ist mit dem Perl-Code die Arbeit des Künstlers getan, sobald die Instruktionen niedergeschrieben und in ihrer Logik fehlerfrei sind. Nicht ein menschlicher Performer, sondern die Maschine führt die Instruktionen aus, so, wie ein mechanisches Klavier eine gestanzte Partitur spielt. Im Kollaps der Grenzen von Architektur und Gebäude, Entwurf und Realisation liegt die künstlerische Faszination des Programmierens. Betrachtet man Konzept, Entwurf und Ausführung als drei Schritte eines Schaffensprozesses – der erste immateriell, der zweite in einer simplifizierten und portablen

Codierung, der dritte in komplexer Codierung[6] – so erübrigt beim Programmieren von Maschinen der zweite Schritt den dritten.

Im Gegensatz zum Binärcode herkömmlicher Daten wie digitalisierter Bilder, Töne und Textdokumente verursacht ein algorithmischer Instruktionscode einen generativen Prozeß. Daß Software Rechner für Berechnungen, nicht bloß als Speicher- und Übertragungsmedium nutzt, unterscheidet sie von nichtalgorithmischem digitalen Daten. Ebenso verhält es sich mit algorithmischen Musikkompositionen gegenüber CD- oder mp3-Tonkonserven, algorithmisch generiertem Text gegenüber sogenanntem „Hypertext“ (einem Datenbank- bzw. Verknüpfungsmodell, das per se keine Algorithmik vorsieht), oder graphischen Hacker-„Demos“ gegenüber Videobändern. Wenn es keine digitalen Daten ohne Programme gibt, die sie speichern, übertragen, modifizieren und an analogen Schnittstellen ausgeben, gibt es auch keine digitalen Künste ohne jene Software-Schichten, die Künstler und ihr Publikum entweder für gegeben halten oder mitgestalten. Zwar kann man Computer nutzen, ohne sie selbst zu programmieren, doch ist es nicht möglich, sie ohne Software zu verwenden. Die Frage ist nur, wer sie programmiert.

**Verknappung des Quellcodes.** Aus den Zufallsversen könnte man schließen, daß der Text zwar variabel ist, seine Partitur jedoch stabil und linear. Dies stimmt zwar und relativiert somit, wie Inke Arns argumentiert, alle Behauptungen der „Nichtlinearität“ oder des Verlusts von Einschreibung in digitalen Kunstwerken,[7]. Trotzdem kann auch ein funktional identisches Programm verschieden notiert werden, auch innerhalb derselben Programmiersprache. Die obigen ersten Quelltexte des Zufallsgedichts zum Beispiel sind zwar einfach zu verstehen, aber unschön formuliert, weil sie Befehlsblöcke redundant wiederholen. Stattdessen könnte man die Anweisung auch so notieren:

(1) Nehmen Sie irgendeine Münze, deren zwei Seiten sich visuell klar voneinander unterscheiden.
(2) Legen Sie fest, welches für Sie die erste und welches die zweite Seite der Münze ist.
(3) Wiederholen Sie die folgenden Anweisungen achtmal:
    (a) Wiederholen Sie die folgenden Anweisungen zweimal:
        (i) Werfen Sie die Münze.

[6] Um eine Dichotomie von Sinnlichem und Abstrakten zu vermeiden, wie sie Nelson Goodman mit seinem Begriffspaar des „autographischen“ und „allographischen“ Zeichens vorschlägt, siehe [Goo76]

[7] Siehe [Arn01]

(ii) Fangen Sie die geworfene Münze so auf, daß sie auf einer ihrer beiden Seite landet.
(iii) Wenn die Münze auf die erste Seite fällt, führen Sie die folgende Anweisung aus:
- Sagen Sie „tumba“

(iv) Anderenfalls führen Sie die folgende Anweisung aus:
- Sagen Sie „ba-umpf“

(b) Machen Sie eine kurze Pause, um das Ende der Zeile anzudeuten.

(4) Machen Sie eine lange Pause, um das Ende des Gedichts anzudeuten.

In Perl würde daraus:

```
for $line (1 .. 8) {
        for $word (1 .. 2) {
                $random_number = int(rand(2));
                if ($random_number == 0) {
                        print "tumba"
                        }
                else {
                        print "ba-umpf"
                        }
                print " "
                }
        print "\n"
        }
```

Noch kürzer und eleganter können die Anweisungen notiert werden, wenn man die Wörter „tumba“ und „ba-umpf“ und die beiden Trennzeichen (Leerzeichen und Zeilenumbruch) jeweils in numerisch indizierte Speicher schreibt, um sie in ein zwei Durchgängen einer Wiederholungsschleife innerhalb der achtfachen Wiederholungsschleife auszulesen:

```
@word = ("tumba","ba-umpf");
@separator = (" ", "\n");
for $line (1 .. 8) {
        for $word_number (0 .. 1) {
                print $word[int(rand(2))], $separator[$word_number];
                }
        }
```

So kollabiert in der Software zwar der Unterschied von Entwurf und Werk, die Verbesserungen des Quellcodes – bei unveränderter Funktion des Programms – zeigen jedoch, daß es auch von algorithmischer Schrift Entwürfe und vorläufige Skizzen gibt, so daß die Tatsache, daß ein Programm gut funktioniert, keine oder nur mittelbare Schlüsse auf die Qualität seines Quellcodes erlaubt. In kommerzieller Softwareentwicklung heißen Software-Entwürfe „rapid prototypes“ und sind Skizzierungen einer Software in einer einfach zu programmierenden Sprache, die aber nur langsam laufende Programme erzeugt; es gibt „Bananensoftware“, die unvollkommen ausgeliefert wird und beim Kunden reift, und „Vaporware“, die nur in der Form einer Produktankündigung existiert, um Aufmerksamkeit auf sich selbst zu lenken und Kunden präventiv von bereits verfügbaren Konkurrenzprodukten abzuhalten. Die Prozessualität von Software drückt sich nicht zuletzt darin aus, daß Computerprogramme mit Versionsnummern versehen werden und Programmierer Versionskontrollsoftware einsetzen, die Änderungen am Code sukzessiv registriert, synchronisiert und gegebenenfalls rückgängig macht.

Auch digitale Kunst und Aufsätze im Netz werden zunehmend mit Versionsnummern versehen. Der Software-Jury des (selbst versionsnumerierten) Festivals transmediale.01 lag das Computerprogramm „Signwave Autoillustrator“ von Adrian Ward – eine Graphiksoftware, deren Werkzeuge ihren äußeren Anschein subvertieren, indem sie algorithmisches Eigenleben entwickeln – in einem Entwicklungsstand vor, den heutige Versionen weit hinter sich gelassen haben und deshalb für interessierte Kuratoren oder Kunsthistoriker nur schwer rekonstruierbar wäre. So gibt es immer noch gute Gründe, digitalen Codes einen „Verlust von Inskription“ zu attestieren, wie es die brasilianische Autorin und Netzkünstlerin Giselle Beiguelman tut.[8] Dieser Verlust unterscheidet sich zwar nicht von der altbekannten Instabilität mündlich überlieferter Literatur sowie musikalischer und theatraler Aufführungen. Neu ist, daß er auch maschinell veröffentlichte und massenreproduzierte Schrift betrifft.

Die gängige programmiertechnische Methode der Eindämmung von Code-Instabilität, ist, verschiedene Funktionseinheiten der Software voneinander zu trennen; so daß nur die Schnittstellen, an denen ein Programmteil einem anderen Daten übermittelt, stabil bleiben müssen, nicht jedoch die Programmteile selbst. Was hier „Schnittstelle“ (oder „API“ für „Application Interface“) heißt, hat wenig mit dem medienwissenschaftlichen Verständnis des „Interface“ als audiovisuell-taktiler Koppelung von Menschen

[8] „Notes on the Loss of Inscription“, `http://www.p0es1s.net/poetics/symposion2001/a{_}beiguelman.html`

und Maschinen zu tun, sondern entspricht in literaturwissenschaftlicher Terminologie einem „Paratext“ wie z.B. der alphabetischen Organisation und Querverweisstruktur eines Lexikons, die als stabile Struktur den instabilen Code von Artikeln verbindet, die über Auflagen hinweg umgeschrieben werden. Bereits die Unterscheidung von „Programmen“ und „Daten“ bezeichnet solche eine Schnittstelle: Ein „digitales Photo“ zum Beispiel ist eine digitale Codierung von Farb-Pixeln, die erst mittels algorithmischer Hilfe des Computerbetriebssystems in elektrische Spannungen umgewandelt wird, die einen Bildschirm oder einen Drucker das Photo zeichnen lassen. Bloße Konvention ist es, nicht die Steueralgorithmen selbst in die digitale Speicherung des Bilds einzubetten, sondern diese Algorithmen in ein „Programm“ oder „Betriebssystem“ zu separieren.[9]

Beispiele computertechnisch normierter Schnittstellen sind Dateiformate, z.B. die Kopfzeilen einer E-Mail-Nachricht, und Netzwerkprotokolle. Wenn Schnittstellen systemübergreifend und herstellerunabhängig genutzt werden sollen, werden sie von Industriegremien wie der IETF (Internet Engineering Taskforce) oder dem W3C (World Wide Web Consortium) analog zu DIN- oder ISO-Normen schriftlich spezifiziert und nicht selten durch eine „Referenzimplementation“ in Gestalt eines Computerprogramms ergänzt, um weitere Implementationen zu ermöglichen. E-Mail zum Beispiel ist im Dokument RFC 822 standardisiert, erlebte seine Referenzimplementation in den 1970er Jahren durch das Unix-Kommando „mail“ und kennt heute zahllose konkurrierende Implementationen in Form von Programmen wie Eudora, Outlook Express oder Mozilla Mail.

Trotz der Veränderbarkeit von Bits und Bytes existieren in der Softwareentwicklung also vielfältige Unterscheidungen von „Entwurf“ und „Werk“, deren Schwelle jedoch nicht ein Wechsel des Materials markiert. Das Material aller Computerprogramme bleiben als Nullen und Einsen abgelegte Algorithmen, ihre Übertragungs- und Speichermedien jedoch wechseln von elektrischem Strom (im Chip) zu magnetischer Ladung (auf der Festplatte oder Diskette), Funkwellen (in drahtlosen Netzen), optischen Markierungen (auf CD- oder DVD-Speichern) oder sogar gebundenem Papier (in Ausdrucken und Handbüchern).

[9] So wäre es denkbar, daß die Medienindustrie, um digitale Privatkopien zu verhindern, demnächst audiovisuelle Daten (wie z.B. Musikaufnahmen) direkt in proprietäre Ein-Chip-Abspielhardware einbetten würde.

## SOFTWAREKUNST

Definiert man Software als ausführbare formale Anweisung oder logische Partitur, dann beschränkt sich sie nicht auf Computer. Die erste, in deutscher Umgangssprache verfaßte Anweisung zur Herstellung des dadaistischen Zufallsgedichts ist nicht minder Software denn ihre drei Übersetzungen in die Programmiersprache Perl, solange ihre Instruktionen gleichermaßen von einem Menschen und von einer Maschine ausgeführt werden können. So wäre auch eine Klavierpartitur dann „Software“, wenn sowohl ein Pianist, als auch ein mechanisches Klavier ihren Code in Musik umsetzen können. Umgekehrt ist auch der Perl-Quellcode des dadaistischen Zufallsgedichts les- und ausführbar, ohne daß man dafür eine Maschine bräuchte. Da jeder Algorithmus bis zu einem kritischen Komplexitätsgrad auch mental ausgeführt werden kann, wie es vor der Erfindung des Computers üblich war, gibt es Software ohne Hardware. Ein Beispiel dafür sind Programmierhandbücher: Der Quellcodes, der in ihnen abgedruckt ist, wird nur selten auf Maschinen ausgeführt, sondern dient dem Leser als Beispiel, dem er intellektuell folgt.[10]

Statt Hugo Balls „Karawane“ in Computerprogramme umzuformen, könnten auch einige historische Dada-Texte, allen voran Tristan Tzaras Anleitung zur Abfassung eines dadaistischen Gedichts durch zufälliges Vertauschen von Wörtern eines Zeitungsartikels, als Software gelesen werden:

> Um ein dadaistisches Gedicht zu machen
> Nehmt eine Zeitung.
> Nehmt Scheren.
> Wählt in dieser Zeitung einen Artikel von der Länge aus, die Ihr Eurem Gedicht zu geben beabsichtigt.
> Schneidet den Artikel aus.
> Schneidet dann sorgfältig jedes Wort dieses Artikels aus und gebt sie in eine Tüte.
> Schüttelt leicht.
> Nehmt dann einen Schnipsel nach dem anderen heraus.
> Schreibt gewissenhaft ab in der Reihenfolge, in der sie aus der Tüte gekommen sind.
> Das Gedicht wird Euch ähneln.
> Und damit seid Ihr ein unendlich origineller Schriftsteller mit einer charmanten, wenn auch von den Leuten unverstandenen Sensibilität.

[10] Was zum Beispiel auch für die Perl-Listings in diesem Text gilt.

Tzaras Anleitung ist ein Algorithmus, Software, die ebenso für einen Computer geschrieben sein könnte.[12] Übersetzte man diese Anleitung aus dem Deutschen oder originalen Französischen in ein Perl- oder C-Programm, würde nicht Kunst in Software umgesetzt, sondern eine nicht-maschinelle in eine maschinelle Softwarekunst transkribiert, vergleichbar etwa mit den formal-kombinatorischen Kompositionsverfahren der seriellen Musik des 20. Jahrhunderts, in denen der Nucleus programmierter Computermusik bereits angelegt war.

Zugegebenermaßen ist der Begriff der Software historisch jünger als die Kunst, auf die er hier rückwirkend angewendet wurde. Was Software ist und in welchem Verhältnis sie zu den zeitgenössischen Künsten steht, ist eine Frage, die sich erstmals der Kunstkritiker und -theoretiker Jack Burnham stellte, als er 1970 eine Ausstellung mit dem Titel „Software“ im New Yorker Jewish Museum kuratierte. „Software“ gilt heute als erste Präsentation der Concept Art. Sie konfrontierte Installationen u.a. von Joseph Kosuth, Art and Language, Hans Haacke und Douglas Huebler mit Computersoftware, die Burnham konzeptuell interessant fand, so zum Beispiel den ersten Prototypen von Ted Nelsons Hypertext-System „Xanadu“.[13] Schon 1961 hatte Henry Flynt Konzeptkunst definiert als Kunst „deren Material ‚Konzepte‘ sind, so, wie z.B. Klang das Material von Musik ist“[14]. Bestimmt man Softwarekunst analog als Kunst, deren Material formaler Instruktionscode ist, so haben beide Künste zwei Merkmale gemeinsam:

(1) die Einheit von Entwurf und seiner Ausführung;
(2) den Gebrauch von Sprache, in Form von Anweisungen in der Softwarekunst und von Konzepten in der Konzeptkunst. Wie Flynt schreibt, stehen „Konzepte in enger Beziehung zur Sprache, so daß Konzeptkunst eine Art der Kunst ist, deren Material die Sprache ist.“[15]

Es ist kein Zufall, daß die meisten zuvor zitierten Beispiele vorelektronischer Softwarekunst literarischer Herkunft sind. Literatur ist eine Konzeptkunst, da sie nicht per definitionem nur an Sprache

[11] Als Teil des „Dada MANIFEST über die schwache Liebe und die bittere Liebe“ publiziert in [Tza78], französisches Original in [Tza75]

[12] Meine Adaption des Texts als Perl-Programm steht unter der Adresse `http://userpage.fu-berlin.de/{~}cantsin/permutations/tzara/poeme{_}dadaiste.cgi` bereit.

[13] Zur Ausstellung „Software“ siehe [Sha]

[14] „of which the material is ‘concepts,’ as the material of for ex. music is sound“, [Fly61]

[15] Ebenda.

> gebunden ist, nicht an an materielle Objekte und Orte. Die Schwierigkeiten des Kunstbetriebs mit digitaler Netzkunst, die sich weder gut ausstellen, noch als Originalobjekt verkaufen läßt, sind dem Literaturbetrieb, der traditionell zwischen Kunstwerk und materiellem Träger unterscheidet, fremd.[16] Da auch formale Sprachen Sprachen sind, ist Software eine Literatur – ohne allerdings den Umkehrschluß. Da formale Sprachen nur eine kleine Untermenge der Sprache bilden, sind Analysen von Softwarecode auf andere Literatur nicht einfach extrapolierbar.

Wenn laut Henry Flynt Konzepte künstlerisches „Material“ werden, dann unterscheidet sich Konzeptkunst von anderer dadurch, daß sie Konzepte exponiert, sich faktisch auf Entwürfe verlegt. Analog unterscheidet sich Softwarekunst von bloß softwaregestützter Kunst dadurch, daß sie ihre Instruktionen und Codiertheit nicht ausblendet. Konzeptkunst im strengen Sinne Flynts und nicht-elektronische Softwarekunst zugleich ist die „Composition 1961“ des amerikanischen Komponisten La Monte Young, dessen Stücke die Anfänge von Fluxus und der Minimal Music prägten. Das Stück besteht aus einer Karte mit der Aufschrift „Draw a straight line and follow it“, eine Anweisung, die unzweideutig genug ist, um auch von einer Maschine ausgeführt werden zu können. Zugleich ist sie nicht konsequent realisierbar, ohne physikalische Grenzen zu sprengen. Also bleibt die Ausführung des Stücks imaginär, konzeptuell.

Entwurf und ausführbarer Code verkoppeln sich auch in Sol LeWitts „Plan for a Concept Art Book“ von 1971, einer Reihe von Buchseiten, die den Leser dazu anweisen, auf ihnen Striche zu ziehen und bestimmte Buchstaben durchzustreichen.[17] Dennoch zeigt sich an LeWitt, daß jene Kunst, die erst in den 1970er Jahren den Begriff „Concept Art“ bekannt machte, in ihrem Konzeptualismus nicht ansatzweise so konsequent und philosophisch radikal war wie jene von Henry Flynt, La Monte Young und Christer Hennix. Während die „Composition 1961“ eine Konzeptbeschreibung eines Kunstwerks ist, das wiederum selbst nur als Konzept mental existieren kann, entwirft LeWitt den „Plan“ eines materiellen, typographisch-visuellen Kunstwerks. So repräsentiert LeWitts Arbeit eine Kunst, die besser „Entwurfskunst“ hieße, weil ihr Material nicht Konzepte sind, sondern in Partituren notierte Bilder und Objekte.

---

[16] Ausnahmen wie visuelle Poesie bestätigen die Regel.

[17] [Hon71], S. 132-140

Hat Programmcode solche Implikationen, so reichen formale Analysen nicht aus. Konzeptkunst bedeutet potentiellen Terror des Konzepts, Softwarekunst potentiellen Terror des Algorithmus, den Terror minimaler Anweisungen also, deren Ausführungen aus dem Ruder laufen. Sades „120 Tage von Sodom“ sind, wie vom Kybernetiker Abraham M. Moles 1971 angedeutet, als eine Programmierung des Exzesses und seiner simultanen Reflexion in Prosa lesbar.[18] Auch der Boom von Spam- und Pseudo-Viren-Codes in zeitgenössischer Digitalkunst bezeugt die perverse Koppelung von Minimalismus und Selbstinflation der Software.[19] An La Monte Youngs „Composition 1961“ können nicht nur die Begriffe von Software und Softwarekunst hinterfragt weredn. Als erste und bis dato eleganteste künstlerische Störsoftware weist sie auch auf die implizite Ästhetik und Politik formaler Instruktionen hin. 1968, zwei Jahre vor Burnhams „Software“-Ausstellung, begründete Donald Knuth mit dem ersten Band seines Buchs „The Art of Computer Programming“ die moderne universitäre Informatik.[20]. Knuths programmatischer Titel schrieb sich in das von Steven Levy überlieferte Hacker-Credo ein, demzufolge „man Kunst und Schönheit mit Computern schaffen“ könne.[21] So recyceln Hacker, sonst die Avantgarde eines umfassenden kulturellen Verständnisses digitaler Technologie, einen klassizistischen Begriff von Kunst als Schönem, und formen daraus ein Konzept digitaler Kunst als innerer Schönheit des Quellcodes; ein ästhetischer Konservativismus, der in ingenieur- und naturwissenschaftlichen Kulturen weit verbreitet ist und neupythagoräischen Digitalkitsch wie z.B. Fraktalgraphiken popularisiert hat.

Softwarekunst durchkreuzt dieses Ideal dann, wenn sie auch die ästhetischen Register des Häßlichen, Monströsen, Dysfunktionalen, Vorgetäuschten und Inkorrekten zieht.[22] Ohnehin entsteht sie nicht mehr in Reinräumen, sondern inmitten einer Abundanz von zirkulierendem Programmcode. Dies unterscheidet sie selbst von Werken wie Tzaras Dada-Gedicht, das zwar einen Überangebot massenmedialer Information voraussetzt, aber nur diese

[18][Mol71], S. 124

[19]Siehe dazu: Franziska Nori (Hrsg.), I love you, Computer Hacker Viren Kultur, Frankfurt a.M. 2002 (Ausstellungskatalog des MAK Frankfurt); die transmediale.02 zeichnete zwei solcher Selbstreplikations-Programme, „tracenoizer“ und „forkbomb.pl“ mit ihrem Softwarekunstpreis aus

[20]knuth:art

[21]nach Steven Levy, [Lev84]. Der deutsche Chaos Computer Club gehört mit seinem jährlichen „Art and Beauty Workshop“ zu den Anhängern dieses Glaubenssatzes.

[22]Weshalb es auch verkehrt wäre, Softwarekunst auf „running code“, also technisch funktionale Algorithmen zu beschränken oder, aus politischen Gründen, Programme mit geheimen Quellcode und unfreier Lizenzierung aus Softwarekunstpräsentationen auszuschließen.

und nicht seinen eigenen Algorithmus collagiert; das Resultat ist chaotisch, der Prozeß jedoch regelhaft.

Seit der Popularisierung von Personal Computern und Internet entsteht Softwarekunst unter der postmodernen Bedingung, daß nicht nur Zeitungsartikel, sondern auch Programmsteuercode massenhaft als Spielmaterial zur Verfügung steht. Die Arbeiten der australischen Netzkünstlerin mez zum Beispiel sind in einer Kunstsprache „mezangelle“ verfaßt, die English mit Code aus Computerprogrammen und Netzwerkprotokollen hybridisiert. Ihre „net.wurks“ sind eine unsaubere, dysfunktionale Softwarekunst; statt Programmcode synthetisch zu konstruieren, verwenden sie die Computercodes als Readymade, sezieren sie und setzen sie so neu zusammen, daß sich semantische Untertöne der vermeintlich bloß technischen Softwaresyntax reflexiv herausschälen. Mit ihr verwandt ist Softwarekunst, die Kontrollparameter kommerzieller Computerprogramme manipuliert. Joan Leandres „retroyou“ und Eldar Karhalevs und Ivan Khimins „Screen Saver“ sind untergründige Mißkonfigurationen kommerzieller Software: eines Autorennspiels, das durch technisch simple, aber wirkungsvolle Eingriffe zu einem Rennen gegenstands- und schwereloser Körper in einem nichteuklidischen Raum wird, sowie des Bildschirmschoners von Microsoft Windows, umkonfiguriert zu einem suprematistischen, bildschirmfüllenden Quadrat.

Wenn eine Kulturwissenschaft und -kritik der Software sich ihren Gegenstand nicht unkritisch aus der Bildschirmgraphik von Desktop-Betriebssystemen heraus erklärt, wird sie um formale Begriffsdefinitionen nicht herumkommen, ohne die es auch kein ästhetisches Bewußtsein poetischer Spekulation und Experimente in Quellcodes geben kann. Durch ihr spielerisches Unterlaufen von Formalismen zeigt die Kunst von mez, Leandre und Karhalev/Khimin jedoch, daß Softwarekunst nicht per se konzeptualistisch ist, sondern sowohl eine Kunst sein kann, deren Material formaler Instruktionscode ist, als auch eine Kunst, die ein kulturelles Verständnis von Software reflektiert.

## LITERATUR

[Arn01] ARNS, Inke: Texte, die (sich) bewegen: zur Performativität von Programmiercodes in der Netzkunst. 2001. – `http://www.v2.nl/~arns/Lecture/performativ-code.html` 6

[Ben62] BENSE, Max: *Theorie der Texte*. Köln : Kiepenheuer und Witsch, 1962 1

[Cag82] CAGE, John: *Roaratorio. Ein irischer Circus über Finnegans Wake*. Königstein/Taunus : Athenäum, 1982 2

[CWM01] COX, Geoff ; WARD, Adrian ; MCLEAN, Alex: The Aesthetics of Generative Code. 2001. – http://www.generative.net/papers/aesthetics/index.html 2

[Fly61] FLYNT, Henry: Concept Art. In: YOUNG, La M. (Hrsg.) ; MACLOW, Jackson (Hrsg.): *An Anthology*. New York : Young and MacLow, 1963 (1961) 11

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

[Goo76] GOODMAN, Nelson: *The languages of art*. Indianapolis / Cambridge : Hackett, 1976 6

[Gra01] GRAU, Oliver: *Virtuelle Kunst in Geschichte und Gegenwart: Visuelle Strategien*. Berlin : Reimer, 2001 1

[Hag97] HAGEN, Wolfgang: Der Stil der Sourcen. Anmerkungen zur Theorie und Geschichte der Programmiersprachen. In: COY, Wolfgang (Hrsg.) ; THOLEN, Georg C. (Hrsg.) ; WARNKE, Martin (Hrsg.): *Hyperkult*. Basel : Stroemfeld, 1997, S. 33–68 1

[Hon71] HONNEF, Klaus (Hrsg.): *Concept Art*. Köln : Phaidon, 1971) 12

[Lan92] LANDOW, George: *Hypertext*. Baltimore : Johns Hopkins University Press, 1992 1

[Lev84] LEVY, Steven: *Hackers*. Champaign, IL : Project Gutenberg, 1986 (1984) 13

[Mol71] MOLES, Abraham A.: *Kunst und Computer*. Köln : DuMont, 1973 (1971) 1, 13

[SB99] SUTER, Beat (Hrsg.) ; BÖHLER, Michael (Hrsg.): *Hyperfiction*. Basel, Frankfurt/M. : Stroemfeld, 1999 (Nexus 50) 1

[Sha] SHANKEN, Edward A.: The House that Jack Built: Jack Burnham's Concept of 'Software' as a Metaphor of Art. In: *Leonardo Electronic Almanach* 6, Nr. 10. – http://www.duke.edu/~giftwrap/House.html 11

[Tza75] TZARA, Tristan: Pour fair une poème dadaïste. In: *Oeuvres complètes*. Paris : Gallimard, 1975 11

[Tza78] TZARA, Tristan: Dada MANIFEST über die schwache Liebe und die bittere Liebe. In: *7 DADA Manifeste*. Hamburg : Nautilus, 1978 11