SpringerWienNewYork

Ästhetik und Naturwissenschaften

Medienkultur

Herausgegeben von

Hans Ulrich Reck

Beirat des Herausgebers

Aleida Assmann (Konstanz)
Karlheinz Barck (Berlin)
Christina von Braun (Berlin)
Horst Bredekamp (Berlin)
Friedrich Kittler (Berlin)
Gertrud Koch (Berlin)
Wolfgang Pircher (Wien)
Christian Reder (Wien)
Sigrid Weigel (Berlin)
Liliane Weissberg (Philadelphia)
Siegfried Zielinski (Köln)

Georg Trogemann
Jochen Viehoff

CodeArt

Eine elementare Einführung in die Programmierung als künstlerische Praktik

**********//

SpringerWienNewYork

Prof. Dr. Georg Trogemann
Dr. Jochen Viehoff
Kunsthochschule für Medien Köln, Deutschland

Die Publikation entstand im Projekt „Informatik, künstlerische Praxis und Kunsttheorie der digitalen Bildtechnologien/KIT“. Das Projekt wurde im Rahmen des Programms der Bund-Länder-Kommission für Bildungsplanung und Forschungsförderung „Kulturelle Bildung im Medienzeitalter – kubim“ (Förderkennzeichen A 6681 A NRW 01) gefördert durch das Bundesministerium für Bildung und Forschung und das Ministerium für Schule, Wissenschaft und Forschung des Landes Nordrhein-Westfalen. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

Das Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere die der Übersetzung, des Nachdruckes, der Entnahme von Abbildungen, der Funksendung, der Wiedergabe auf photomechanischem oder ähnlichem Wege und der Speicherung in Datenverarbeitungsanlagen, bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten.
Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Buch berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

© 2005 Springer-Verlag/Wien
Printed in Austria
SpringerWienNewYork ist ein Unternehmen von Springer Science+Business Media
springer.at

Graphische Gestaltung: Rajele Jain
Druck und Bindearbeiten: Grasl Druck & Neue Medien, 2540 Bad Vöslau, Österreich
Gedruckt auf säurefreiem, chlorfrei gebleichtem Papier – TCF

SPIN: 10969410
Mit zahlreichen, zum Teil farbigen Abbildungen

Bibliografische Information Der Deutschen Bibliothek: Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

ISSN 1430-5321
ISBN 3-211-20438-5 SpringerWienNewYork

INHALTSVERZEICHNIS

VORWORT 9

KAPITEL 1 DIE PROGRAMMIERBARE MASCHINE
Georg Trogemann

Die programmierbare Maschine 17
1.1 Der Computer als Artefakt 21
1.1.1 Zum Wesen der Technik 23
1.1.2 Programme als Realitätskonstruktionen 30
1.1.3 Programmieren im künstlerischen Kontext – eine erste Annäherung 37

1.2 Die Struktur der programmierbaren Maschine 47
1.2.1 Unsichtbare Maschinen 49
1.2.2 Automaten und mechanische Rechenmaschinen 54
1.2.3 Die rationalistische Tradition 62
1.2.4 Formalisierung 65
1.2.5 Programmierbarkeit im engeren Sinn 72
1.2.5.1 Analoge Programme 73
1.2.5.2 Programme als Code 74
1.2.5.3 Steckfeld-Programmierung 76
1.2.5.4 Maschinen mit Programmspeicher 78
1.2.5.5 Universelle Kontrollstrukturen 79
1.2.5.6 Die universell-programmierbare Maschine 82

1.3 Die hybride Maschine 87
1.3.1 Code-Reflexivitäten 88
1.3.1.1 Selbstreferenz und Kybernetik 2-ter Ordnung 89
1.3.1.2 Computational Reflection – maschinelle Selbstbetrachtungen 91
1.3.1.3 Metasysteme 95
1.3.1.4 Heterarchien 99
1.3.2 Interfaces 107
1.3.2.1 Interaktivität & Interaktion 108
1.3.2.2 Interface-Mapping 110
1.3.2.3 Die Zukunft des Interfaces 114
1.3.3 Flottierende Codes 116

1.4 Konsequenzen für die künstlerische und kulturelle Praxis der neuen Medien 119
1.4.1 Programmierung als Zeichenprozess 120
1.4.2 Medienobjekte 125
1.4.2.1 Hierarchische Organisation von Daten 126
1.4.2.2 Alphanumerische Repräsentation & Datenabstraktion 128
1.4.2.3 Datennetze 131
1.4.2.4 Die Prozessstruktur medialer Objekte 132
1.4.2.5 Medienobjekte und Code-Reflexivitäten 133
1.4.3 Interfaces – Zwischen Wahrnehmung und Kognition 135
1.4.3.1 Tasten, Schalter, Regler 136
1.4.3.2 Komplexe Synthesen und Analysen 138
1.4.4 Zum Verhältnis von Autor und Werk 141
1.4.5 Offenheit – Wesentliches bleibt unformalisiert 146
1.4.5.1 Abstraktion 148
1.4.5.2 Verschränkung paralleler Aktivitäten – Heterarchien 149
1.4.5.3 Die Unschärfe des Interfaces 152
1.4.6 Abschließendes 153

KAPITEL 2 GRUNDLEGENDE STRUKTUREN DER PROGRAMMIERUNG
- Zwischenspiele
Georg Trogemann

Programmierparadigmen und die Entstehung von Java 159
Programmierparadigmen / Höhere Programmiersprachen / Eine kurze Geschichte der Programmiersprache Java

Freie Software & Black Box 170
Freie Software und offene Quellen / Black Box – White Box / Künstlerische Programmierung – Zwischen freier Software und Black-Box-Denken

Algorithmisches Denken 180
Endlichkeit / Elementarität / Determiniertheit / Unbeschränktheit / Allgemeinheit / Materialunabhängigkeit / Theorie der Berechenbarkeit / Was ist ein ‘effektives Verfahren’? / Was ist ein ‘Problem’?

Komplexität – Abstraktionen, Module und Hierarchien 189
Objektorientierte Strukturierung von Software-Systemen / Abstraktion / Kapselung / Hierarchisierung / Skalenhierarchie / Kontrollhierarchie / Abstraktionshierarchie / Vererbung

Elektroakustische Musik – algorithmische Komposition 207
Sequenzer / MIDI / Algorithmisches Komponieren

Von den indischen Zahlen zur Leibnizschen Dyadik 222
Die indischen Zahlen / Stellenwertsysteme / Die Leibnizsche Dyadik

Die Rechenmeister und die Cossisten 234
Rechenschulen / Der Methodenstreit / Die Cossisten

Universelle Turingmaschinen & minimale Kontrollstrukturen 244
Die universelle Turingmaschine / Stored Program Electronic Computers / Universelle Kontrollstrukturen / Eine minimale universelle Programmiersprache

Das Computerbild – Oder: Die Janusköpfigkeit des Pixels 263
Mathematische Bilder / Die mathematische Beschreibung kontinuierlicher Bilder / Die mathematische Beschreibung diskreter Bilder / Das Computerbild /

Interfaces – Von Menschen und Mäusen 289
Tracking – Die Technologien der Verfolgung / Das Problem der Kontrolle / Augmenting Human Intellect

Animation – Vom Thaumatrop über das Kino zum algorithmischen Bewegtbild 304
Visuelle Persistenz / Die Entwicklung der Filmtechnik / Das algorithmische Bild

Interaktive Erzählformen 320
Erzählstrukturen / Antizipation / Kontrolle versus Autonomie

GRUNDLEGENDE STRUKTUREN DER PROGRAMMIERUNG
Jochen Viehoff

2.1 Die objektorientierte Programmiersprache Java 165
2.1.1 Java-Editionen und die Java 2 Plattform 165
2.1.2 Java-Compiler und Byte-Code 167

2.2 Das CodeKit-Programmarchiv 176
2.2.1 Nutzung und Installation 176
2.2.2 Medienobjekte 177
2.2.3 Kontakt zur Außenwelt 178

2.3 Getting Started 187
2.3.1 Video-Loop und die Endlosschleife 187

2.4 Objekte und Klassen in Java 196
2.4.1 Kommentare und Blöcke in Java-Klassen 196
2.4.2 Importware 197
2.4.3 Eine leere Java-Klasse 198
2.4.4 Allgemeiner Aufbau von Klassen: Methoden und Eigenschaften .. 199
2.4.5 Ausführbare Objekte I: Vererbung und Java-Applets 200
2.4.6 Instanzen einer Java-Klasse 202
2.4.7 Ausführbare Objekte II: Java-Applikationen 204
2.4.8 Konstruktor und Selbstinstanziierung 205

2.5 Die Kunst der Fuge 215
2.5.1 Das CodeKit-Medienobjekt `MidiInstrument` 215
2.5.2 Java-Applet `FirstMelody` spielt Bach 218

2.6 Elementare Datenstrukturen 228
2.6.1 Primitive Datentypen 228
2.6.2 Lokale und globale Variablen 230
2.6.3 Listen, Felder und Indizes 230

2.7 Elementare Operatoren 240
2.7.1 Arithmetische, logische und vergleichende Operatoren 240
2.7.2 Mathematische Funktionen in Java 242

2.8 Elementare Kontrollstrukturen 255
2.8.1 Schleifenkonstruktionen in Java 255
2.8.2 Bedingte Verzweigungen 260

2.9 Digitale Bilder 271
2.9.1 Koordinatensysteme 271
2.9.2 Darstellung grafischer Objekte 271
2.9.3 Laden und Anzeigen digitaler Bilder in Java-Applets 275
2.9.4 Bildschirmfenster in Java-Applikationen 277
2.9.5 RGB-Farbkodierung für digitale Bilder 278

2.9.6 Digitale Bildbearbeitung I – Spiegelung und Drehung 283
2.9.7 Digitale Bildbearbeitung II – Einfache Filter 286

2.10 Standard-Interaktionen in Java 296
2.10.1 Event-Steuerung in Java-Programmen 296
2.10.2 Einfache Schnittstellen I – Mouse-Interaktion in Java 297
2.10.3 Einfache Schnittstellen II – Tastatur-Eingaben in Java 301

2.11 Eigenständige Prozesse in Java: Threads und Animationen 310
2.11.1 Starten eines Threads 310
2.11.2 Die Endlosschleife `while(true)` 312
2.11.3 Bewegte Objekte und Bild-Animation 314

2.12 Interaktion und Steuerung audio-visueller Medien 326
2.12.1 Abspielen und Steuern von Video-Sequenzen 326
2.12.2 Wiedergabe und Steuerung von Audio-Dateien 329
2.12.3 Drag & Drop – Ein Geduldsspiel 336
2.12.4 Temperaturabhängige Steuerung eines Videos 339

KAPITEL 3 FORTGESCHRITTENE PROGRAMMIERTECHNIKEN
- Zwischenspiele
Georg Trogemann

Zufall & Determinismus 345
Vorhersagbarkeit / Kausalität / Simulierter Zufall

Analytische Geometrie & Zentralperspektive 372
Metrischer Raum / Zentralperspektive

Die Weltmaschine 391
Billard-Computer / Partikelsysteme

Feedback & Chaos 415
Deterministisches Chaos / Wachstumsprozesse / Die logistische Funktion

Zelluläre Automaten & Selbstreproduktion 440
Zelluläre Automaten und universelle Konstruktoren / Selbstreproduktion

Grammatiken & Automaten 459
Zum Begriff des 'Automaten' / Grammatiken

Digitaler Darwinismus 482
Homunculus und Robot / Die Evolution des blinden Uhrmachers / Evolutionsstrategien, genetische Algorithmen und genetische Programmierung / Genetischer Basis-Algorithmus

Logische Kalküle, Perzeptren und neuronale Netze 522
Zur Geschichte der Gleichsetzung von Gehirn und Maschine / Konnektionismus versus Symbolverarbeitung / Das Perzeptron / Künstlich versus natürlich

Denkende Maschinen 545
Künstliche Intelligenz als Forschungsfeld / Das Spiel NIM / Kategorien des Lernens

Die ultimative Maschine 564
Erewhon / Der Bio-Adapter / Cyborgs

FORTGESCHRITTENE PROGRAMMIKERTECHNIKEN
Jochen Viehoff

3.1 Zufallszahlen 357
3.1.1 Erzeugung von Zufallszahlen 357
3.1.2 Korrelationen: Gute und schlechte Zufallszahlen 360
3.1.3 Farbrauschen: Der Java-Zufallszahlengenerator 364

3.2 Dreidimensionale Grafik 378
3.3.1 Koordinaten, Translationen und Rotationen 378
3.3.2 Projektionen und Perspektiven 383

3.3 Partikelanimation und klassische Mechanik 400
3.3.1 Massen, Vektoren und Kräfte 400
3.3.2 Numerische Integration der Bewegungsgleichungen 403
3.3.3 Zweikörperanimation 407
3.3.4 Exakte Integration: Das Apfel-Experiment 409
3.3.5 Vielteilchensysteme: LaLinea und BitMirror 410

3.4 Digitale Rückkopplungen und Fraktale 422
3.4.1 Iterative Funktionen und das Feigenbaum-Diagramm 422
3.4.2 Darstellung der Mandelbrotmenge 427
3.4.3 Digitales Video-Feedback 432
3.4.4 Rechenfehler und digitale Ungenauigkeiten 437

3.5 Zelluläre Automaten 447
3.5.1 Eindimensionale Automaten 447
3.5.2 Der Klassiker: Conways Game of Life 453

3.6 Wachstums-Algorithmen 470
3.6.1 Funktionensysteme: Der Barnsley-Farn 470
3.6.2 L-Grammatiken im Blätterwald 475

3.7 Genetische Algorithmen 509
3.7.1 Gene, Fortpflanzung und Mutationen – Parametersuche für den eindimensionalen zellulären Automaten 509
3.7.2 Das interaktive Java-Applet `GeneticAutomata` 514

3.8 Neuronale Netze 532
3.8.1 Das Perzeptron in Java 532
3.8.2 Mustererkennung als Lernprozess 540

3.9 Künstliche Intelligenz 555
3.9.1 Das Nimm-Spiel in Java 555

Projekt-Steckbriefe 493

ANHANG

Literaturverzeichnis 573

Java-Glossar 583

Befehlsübersicht der Java-Programmiersprache 589

Abbildungsnachweis 593

VORWORT

„Die Welt, die wir bewohnen ist eine technische Welt."[1] Gleichwohl bleibt unser Verhältnis zur Technik erschreckend unreflektiert, und bei den meisten Menschen dominiert ein Gefühl der Ohnmacht gegenüber dem permanenten Strom neuer technischer Artefakte. Hier wurzelt sicherlich eines der großen Probleme unserer Zeit, unsere offensichtliche Unfähigkeit, „die Technik geistig in der Hand zu halten."[2] Technikentwicklung ist gekennzeichnet von einer enormen Eigendynamik, die alles ausschaltet, das sich nicht direkt einem oberflächlichen Innovationsverständnis oder dem schnellen kommerziellen Erfolg unterordnet. Aufgrund dieser ungebremsten Kraft läuft die Technik der geistigen Reflexion immer voraus und erzeugt damit die gefährliche Differenz zwischen Technik und Lebenswelt. Es ermangelt uns bisher eines Bildungsbegriffs, der gleichermaßen auf technische Kenntnisse und technisches Handeln, wie auch auf unsere geistige Existenz abzielt. In diesem Zusammenhang stellt sich auch die Frage, wie kulturelle Bildung auf dem Gebiet der fortgeschrittenen Informations- und Kommunikationstechnologien – einem der Schlüsselbereiche unserer technologischen Zivilisation – aussehen kann. Konsensfähig ist sicherlich die Forderung, dass es in der künstlerischen wie der wissenschaftlichen Bildung nicht um oberflächliches technisches Wissen gehen kann, das mit jeder neuen Hard- oder Softwaregeneration veraltet. Vielmehr muss die Vermittlung allgemeiner Prinzipien im Zentrum stehen, die sich nicht nur in langen Zeiträumen wandeln, sondern die vor allem in unserer geistigen Existenz wurzeln. Welche allgemeinen Erkenntnisse sollen wir nun aber aus einem Wissen über die Grundstrukturen computerbasierter Medien ziehen? Weshalb sollte eine derart spezielle Tätigkeit wie das Schreiben von Computerprogrammen von allgemeinem Interesse sein?

Zuallererst bedeutet Programmieren, Maschinen zu konstruieren. Gleichzeitig ist es ein Weg, die Logik des Computers zu erfahren und sich das eigene Verhältnis hierzu deutlich zu machen. Doch dies ist vordergründig und bleibt in der Maschine selbst verhaftet. Sehr viel wichtiger ist dagegen, dass über das Verständnis algorithmischer Grundstrukturen allgemeine Einsichten in die Prinzipien, Möglichkeiten und Grenzen der Mechanisierung und des mechanistischen Weltbildes gewonnen werden können, das heute als gleichermaßen kritisiertes wie verteidigtes Erklärungsprinzip für nahezu alle natürlichen wie künstlichen Erscheinungen herangezogen wird. Mechanisierung ist quasi das Endergebnis einer rationalen Einstellung zur Welt und die programmierbare Universalmaschine ist ihre materielle Ausstülpung. Vor diesem Hintergrund verstehen wir im vorliegenden Buch Maschinen als materialisierte Projektionen von Wesensmerkmalen des Menschen und die Organisationsprinzipien von Computergrogrammen als Ablagerungen sozialer Beziehungen. Da die technisch entäußerten Denkstrukturen

1 Max Bense, Technische Existenz, S. 122.

2 Ebd., S. 124.

uns als eigenständige Objekte wieder begegnen und auf uns zurückwirken, ist der Computer eine evokatorische Maschine, die uns vor allem beim Programmieren unser eigenes Denken erkennen lässt. Dieser wichtige Punkt wird im Laufe des Buches an verschiedenen Stellen herausgearbeitet.

Doch dies ist bestenfalls die halbe Wahrheit über die programmierbare Maschine und ihre gesellschaftliche Bedeutung. Ein wesentliches Merkmal unterscheidet Computer von herkömmlichen Maschinen. Als Werkzeuge und Artefakte zur Steuerung von Produktions- und Arbeitsprozessen gehören Computer zum Bereich der Technik. Als Systeme, die Informationen speichern, darstellen und übertragen, begegnen sie uns aber als Medien. Unser derzeitiges Bild vom Computer changiert zwischen Technik und Medium. Zunächst konzipiert als Hilfsmittel zur Befreiung des Menschen von Routineaufgaben des Geistes erledigten Computer seit Mitte des 20. Jahrhunderts schnell und zuverlässig riesige Stapel repetitiver Rechenaufgaben. Von hier ausgehend entwickelten sie sich aber rasch zu universellen Werkzeugen der Modellierung und Bearbeitung beliebiger Daten. Gegen Ende des 20. Jahrhunderts erfolgte dann eine zweite Ausweitung. Computer wurden zu Knoten eines globalen Netzes, dem das Potential zugesprochen wird, die gesellschaftlichen Kommunikations- und Wissensstrukturen nachhaltig zu verändern. Doch die technischen Medien erzeugen nicht von sich aus neue kulturelle Formen. Ihr Potenzial kann nur in kreativen Prozessen jenseits der informatischen Ebene der Speicherung, Übertragung und Verarbeitung von Informationseinheiten entfaltet werden. Mit dem Erscheinen des Interfaces als Vermittlungsinstanz zwischen den logischen Strukturen der Maschine und den Wahrnehmungsprozessen ihrer Benutzer dringen die Anwendungen zudem mit Vehemenz in kulturell und sozial dominierte Lebensbereiche vor, für deren Behandlung die bisherige technische Ausbildung vollkommen unzureichend ist. Durch die algorithmische Analyse und Synthese von Bildern, Klängen und Handlungen öffnet sich ein neuer konzeptueller und experimenteller Raum des Zusammenspiels von Kognition und Wahrnehmung.

All das mag noch nicht als Begründung reichen, warum sich Künstler im eigenen Interesse mit dem Medium Computer auseinandersetzen sollten. Vielmehr macht das Gesagte deutlich, dass umgekehrt die neuen Medien zwingend auf die Kunst und die Kultur als notwendige Ergänzungen zu den technisch geprägten Veränderungen angewiesen sind. Kunst und Kultur stellen Reflexionsformen und Handlungsalternativen bereit, die den Natur- und Technikwissenschaften nicht zur Verfügung stehen, nur so kann das inhaltliche Potenzial dieser Technologien ausgeschöpft werden. Warum aber sollten sich Künstler auf die technische Logik der Maschine und die eingleisigen Denkstrukturen stringenter Softwaresysteme einlassen? Einerseits hat eine kleine künstlerische Avantgarde den neuen konzeptuellen Raum, den die algorithmische Synthese und Analyse eröffnen, als spannendes Arbeitsfeld für sich entdeckt. Doch eine ausgreifendere Antwort, die auch die scheinbaren Verweigerer mit einbezieht, ist folgende: Die Kunst hat sich längst

auf die Konsequenzen der neuen Medien eingelassen, sie hatte auch nie eine Chance, sich dem zu entziehen. Die Vorstellung, dass Kunst von Alltagseinflüssen getrennt werden kann, ist eine ähnliche Idealisierung wie die Schimäre der Objektivität in den exakten Wissenschaften. So wird nicht nur in allen Bereichen der traditionellen Kunst mit Computern gearbeitet, die Auswirkungen der Technik und die Ästhetiken der Computeroberflächen spiegeln sich selbst in Arbeiten der traditionellen Malerei wider. Dabei allerdings bleiben die eigentlichen Strukturen der Maschine meist verborgen. Hier gilt es nun anzusetzen. „Softwarekunst ist keine Verfahrenstechnik zur Erzeugung anziehender Kunstoberflächen. Sie ist der realistische künstlerische Zugriff auf Software als allgegenwärtiges Material unserer Gesellschaft. Der unausgesprochene Vorwurf der Softwarekunst an die Kunst betrifft deshalb keinen Wettstreit innovativer Formen, sondern einen blinden Fleck der Kunst. Denn was kann Kunst können, wenn sie die informatischen, genetischen, ökonomischen Codesprachen nicht beherrscht? ... 'Software Art' ist eine Reportage über den Code. Denn Softwarekunst ist keinesfalls Kunst für Programmierer. Softwarekunst ist die Lesbarmachung des Codes."[3] Diese Charakterisierung des avancierten Feldes der 'Software Art' spiegelt auch ein wesentliches Anliegen unseres Buches wider. **CODE@ART** will der Kunst den Zugang zu den Codesprachen des Computers ermöglichen. Unser Ziel ist dabei nicht, direkt und umfassend zu künstlerischen Fragen und Strategien der Auseinandersetzung mit Programmcodes Stellung zu nehmen, sondern vielmehr die Grundprinzipien der Programmierung aufzudecken und damit Software überhaupt erst als Reflexionsgegenstand verfügbar zu machen. Weiterhin gehört es nicht zu unseren Absichten, Direktiven, Folgeabschätzungen oder gar Zukunftsszenarien anzubieten, sondern zuerst gilt es, die wesentlichen Komponenten der programmierten Medien besser zu verstehen. Wir begreifen Programme als kulturell und gesellschaftlich codierte Mittel, die eingebettet sind in kommerzielle wie politische Rahmenbedingungen. Gleichviel, ob man in Zukunft für oder gegen die computerbasierten Technologien ins Feld ziehen will, es ist anzuraten, sich vorher mit ihrer Struktur vertraut zu machen.

In der Praxis fortgeschrittener Computertechnologien überlagern sich soziale und künstlerische Strategien und Wahrnehmungskonventionen mit den kognitiven Strukturen des mechanistisch-rationalistischen Denkens. Von dieser Konfrontation unterschiedlicher Denk- und Reflexionskulturen, die inzwischen in vielfältigen neuen Medienformen ihren Ausdruck finden, haben wir uns bislang nicht erholt. Es gilt nun das, was inzwischen vielerorts tägliche Praxis ist, aufzuarbeiten und daraufhin zu untersuchen, welche immanenten Strukturen wirksam sind und welche Potentiale und Beschränkungen sich daraus ableiten. Mit der Erweiterung kultureller Formen ändern sich nicht nur unsere bisherigen Wahrnehmungsgewohnheiten, es verändern sich auch die Arbeitsstrukturen und ihre Anforderungen. Im künstlerischen Kontext ist der Computer nicht mehr länger ausschließlich Werkzeug der Objektivierung

3 Gerrit Gohlke, in: Künstlerhaus Bethanien (Hrsg.), Software Art, S. 4.

und Standardisierung, sondern zunehmend Mittel des subjektiven und individuellen Ausdrucks. Programmierende Künstler begreifen den Code als Material und damit wesentliche Komponente ihrer Projekte. Künstlerische Software ist hierbei nie ganz fertig, sie bleibt stets unscharf, offen und spekulativ. Sie enthält immer die Aufforderung, weiter über sie nachzudenken und sie zu verändern. Damit jedoch die Erforschung neuer Ausdrucksmöglichkeiten gelingt, ist im Umgang mit dem Computer der traditionellen Schule der Wahrnehmung mit allen Sinnen eine Schule der Kognition an die Seite zu stellen.

Virtuos programmieren zu können, stellt aber noch nicht sicher, dass man auch eine tieferes Verständnis vom Wesen dieser Tätigkeit besitzt. Bekanntlich kann man sich leichter auf eine Sache verstehen, als die Sache selbst zu verstehen. Die theoretische und ästhetisch-praktische Reflexion des Programmierens geschieht jedoch nicht von selbst, sondern bedarf eigener Methoden und Strategien der Analyse, des Experiments und der individuellen Auseinandersetzung. In dieser Hinsicht ist das Buch auch für Informatiker gedacht. Die allermeisten Einführungen in die Programmierung konzentrieren sich auf die reine Fertigkeit der Codierung und glauben, damit sei schon alles Notwendige gesagt. Doch Softwarestrukturen sind keine technische Errungenschaft der letzten 50 Jahre, sondern stehen in einer kontinuierlichen Denktradition, die so alt ist wie die Menschheit selbst. Auch die Geisteswissenschaften – allen voran die jüngere Medienwissenschaft – sind dazu übergegangen, nach den medialen Bedingungen von Literatur, Kultur und Kunst zu fragen. Soll aber die Struktur und das Potenzial computerbasierter Medienformen wissenschaftlich untersucht werden, sind neben Kenntnissen der technischen und historischen Fakten des Mediums Computer das Wissen um die Bedingungen seiner Programmierbarkeit unverzichtbar. Im Zentrum des bisher wenig untersuchen Verhältnisses von künstlerischer Praxis, Informatik und Medientheorie steht die programmierbare Maschine. Die Sprachen der neuen Medien sind Programmiersprachen.

Programmierung wird im vorliegenden Buch als zukünftige elementare Kulturtechnik verstanden, die in einer faszinierenden und Jahrhunderte zurückreichenden Denktradition steht. Das Buch wendet sich an Leser, die einerseits Programmieren anhand einschlägiger Beispiele aus den Bereichen Bild, Klang und Interaktion für die objektorientierte Sprache Java lernen wollen, und die andererseits an den spannenden kultur- und technikgeschichtlichen Hintergründen der Programmierung interessiert sind. CodeArt ist damit die erste Einführung in die Programmierung, die insbesondere Medienkünstler/Gestalter, Medieninformatiker und interessierte Geisteswissenschaftler anspricht. Um dieser Zielsetzung des Buches gerecht zu werden, wurde ein mehrschichtiger Aufbau gewählt. Das einführende Kapitel stellt zunächst die Strukturen und Grundprinzipien der programmierbaren Maschine vor und diskutiert verschiedene Konsequenzen, die sich hieraus für die künstlerische Praxis mit computerbasierten Medien ergeben. Diese Analyse bildet das gedankliche Gerüst für die zweite Ebene, den eigentlichen Programmierkurs,

der das 2. und 3. Kapitel des Buches umfasst. Als Programmiersprache dient hierbei die objektorientierte und plattformunabhängige Sprache Java. Im 2. Kapitel werden zunächst die elementaren Strukturen der Programmierung aus medienzentrierter Sicht dargestellt. Im Mittelpunkt stehen elementare mediale Objekte aus den Bereichen Bild, Klang und Interfaces. Grundlegende Daten- und Kontrollstrukturen werden an praxisrelevanten Beispielen eingeführt. Das 3. Kapitel widmet sich dann verschiedenen fortgeschrittenen Methoden, die in der künstlerischen Praxis mit neuen Medien von besonderem Interesse sind. Ausgewählt wurde eine Reihe von Standardmethoden, die in einer Vielzahl künstlerischer Arbeiten ihren Niederschlag gefunden haben. Das Interesse der Kunst an gerade diesen Methoden hat vermutlich mit ihrer Eigenschaft zu tun, dass die resultierende Software eine gewisse Offenheit und Autonomie zeigt, zumindest wird dies oft von den Autoren als wesentliches Merkmal ihrer Arbeiten herausgestellt. Gerne hätten wir eine Reihe weiterer interessanter – jedoch relativ komplizierter – medialer Methoden aufgenommen, beispielsweise Datenbankstrukturen, kollaborative Filter, statistische Verfahren, oder Suchmaschinen. Aus Platzgründen musste eine enge Auswahl an Methoden getroffen werden. Eine dritte Einheit des Buches wird durch eine Reihe von Zwischenspielen gebildet, die einzelne Begriffe und Themenfelder tiefer gehend beleuchten. Im 2. Kapitel sind diese Zwischenspiele in erster Linie historische Anmerkungen zu den Grundelementen der Programmierung. Die Zwischenspiele des 3. Kapitels lehnen sich stark an die dort vorgestellten weiterführenden Methoden an und versuchen auf knappem Raum sowohl historische Hinweise auf den Ursprung dieser Methoden zu geben, als auch ihre wesentlichen Grundprinzipien zu beschreiben. Leser, für die das Erlernen der Programmierung im Zentrum steht, können zunächst direkt mit den Kapiteln 2 und 3 beginnen, die eine in sich abgeschlossene Einführung in die Java Programmierung bieten. Wer sich dagegen für die kultur- und technikgeschichtlichen Hintergründe der Programmierung interessiert, erhält in Kapitel 1 zusammen mit den 22 Zwischenspielen eine umfassende Einführung in die Geistesgeschichte der programmierbaren Maschine. Eine wichtige Ergänzung zum Buch ist die Webseite (`http://java.khm.de/`) auf der alle Programmbeispiele und weitere Java-Applikationen und Applets frei verfügbar sind. Während das Buch anstrebt, Grundlegendes zu vermitteln, soll die Webseite vor allem das praktische Erlernen der Programmiersprache Java unterstützen. Buch und Webseite zusammen bilden nicht nur eine andersartige Einführung in die Programmierung, sondern können gleichermaßen als Grundlage und Quelle für die Realisierung eigener Projekte dienen. Die Source Codes bilden zusammen eine Art Baukastensystem, das frei weiterverwendbar ist und ohne Einschränkungen an eigene Projektbedürfnisse angepasst und in neuen Kontexten eingesetzt werden kann.

Köln, September 2004

KAPITEL 1

DIE PROGRAMMIERBARE MASCHINE

Die programmierbare Maschine

„Why have we become technological gods and moral devils, scientific superman and esthetic idiots? ... An age that has devaluated all its symbols has turned the machine itself into a universal symbol: a god to be worshiped. Under these conditions, neither art nor technics is in a healthy state."
Lewis Mumford[1]

„Die 'hard sciences' sind erfolgreich, weil sie sich mit den 'soft problems' beschäftigen; die 'soft sciences' haben zu kämpfen, denn sie haben es mit den 'hard problems' zu tun."
Heinz v. Foerster[2]

Seit jeher haftet den Ingenieuren der Verdacht der Kulturlosigkeit an. Dies ist auch auf dem Gebiet der Computertechnologien nicht grundlegend anders, obwohl hier neben mathematisch-technischen Fragen zunehmend kulturelle Inhalte ins Zentrum der Betrachtungen und Anwendungen rücken. Das Phänomen der scheinbar ungeistigen Technik ist beileibe nicht neu. „Die Antike und das Mittelalter betrachteten Technik als eine nur handwerksmäßig ausgeübte Kunstfertigkeit, die einer theoretischen Untersuchung weder fähig war, noch würdig ist. Verglichen mit dem Ideal theoretischer Argumente, folgerichtiger Ableitungen und allgemeiner Gesetzmäßigkeiten wurden das handwerkliche technische Können und der praktische Vollzug technischer Handlungen eher als Gegenstände minderen Ranges angesehen."[3] Unter τεχνη (techné) verstanden die Griechen primär diejenigen Fertigkeiten und Geschicklichkeiten, die bestimmte Leistungen und Produkte hervorzubringen vermochten und die man durch Abschauen und Nachmachen erlernen konnte. „Daß man eine Technik in diesem Sinne erlernen konnte, daß man sich auf die Sache verstehen konnte, ohne die Sache selbst zu verstehen und die Notwendigkeit der Verrichtung auf das Wesen dieser Sache zurückführen zu können, unterschied das technische Sich-Verstehen-auf von dem theoretisch-wissenschaftlichen Verhältnis zum Gegenstand."[4] Die Trennung von Sachverstand und Sachbeherrschung wurde von den Griechen aber dennoch als große Chance verstanden und Technik und Wissen, Fertigkeit und Einsicht wurden noch auf eine gemeinsame Wurzel zurückgeführt, denn letztlich musste „die Tradition des Erlernens und Nachmachens doch zurückgehen auf einen, der die Fertigkeit aus der Einsicht ein für allemal gewonnen hatte, der das der Sache Angemessene aus der Erkenntnis dieser Sache zu entwickeln gewusst hatte".[5]

1 In: Art and Technics, S. 139.
2 In: Short Cuts, Kompetenz und Verantwortung, S. 183.
3 Friedrich Rapp, Die Dynamik der modernen Welt: Eine Einführung in die Technikphilosophie, S. 13.
4 Hans Blumenberg, Wirklichkeiten in denen wir leben, S. 12.
5 Ebd.

Der Makel des Ungeistigen der Technik wurde bis in unsere Zeit fortgeschrieben. In seinem 1929 erschienenen Buch *Die Technik als Kulturproblem* beklagt der an der Technischen Hochschule München lehrende Kunsthistoriker und Kunsttheoretiker Josef Popp, dass „die zukünftigen Techniker sich ihres Berufes im Rahmen der allgemeinen und persönlichen Kultur nicht genügend bewusst sind“ und „in ungleich geringerem Maße als andere Berufsanfänger verstehen, ihren Beruf geistig zu nehmen.“[6] Im zwanzigsten Jahrhundert werden nicht mehr handwerklich-technische Praxis und reflektierende Theorie getrennt, sondern der Riss zieht sich durch die Theorie selbst. Trotz aller Beschwörung interdisziplinärer Netzwerke und fächerübergreifender Kooperationen gilt heute nach wie vor C.P. Snows Unterteilung der wissenschaftlichen Gesellschaft in zwei Kulturen: die Technik- und Naturwissenschaftler und die (literarischen) Intellektuellen. Die Kluft zwischen den beiden Kulturen wird immer dann besonders deutlich, wenn gesellschaftliche Gefahren, Nebenwirkungen und Spätfolgen der Technik diskutiert werden. Eine gute Analyse des heutigen Problems der Technik liefert David Köpf,[7] der bei seiner Argumentation vor allem auf die Arbeiten des Philosophen Gotthard Günther zurückgreift. Bei der systemtheoretischen Darstellung des Problems bezieht er sich allerdings an einer Stelle auf die zugespitzte Formulierung des Problems durch Luhmann:[8] „Im gesamtgesellschaftlichen Kontext gesehen, haben die Naturwissenschaften ihre Effekte dadurch, daß sie das Handeln und nur über das Handeln das Erleben der Menschen beeinflussen, während für die Humanwissenschaften das Umgekehrte gilt: daß sie primär das Erleben und nur über das Erleben das Handeln beeinflussen. Anders als die sog. Praktische Philosophie es sieht, provoziert die Gesellschaft sich gerade durch die Naturwissenschaften zum Handeln, weil hier die Wahrheiten genügend eindeutig sind, um Handlungsanschlüsse mit abzudecken. Im Bereich der Humanwissenschaften ist genau dies dagegen typisch ungesichert, die gesellschaftlichen Auswirkungen sind hier daher viel stärker über Plausibilitäten in Bezug auf richtiges Erleben vermittelt, und das heißt auch anfälliger gegen Ideenabnutzung, Ideenwandel, Ideenpolitik, ... Jedenfalls ist diese Differenz ein Faktum, das eine eigentümliche Schieflage produziert und reproduziert, und zwar gerade wegen des eigentümlichen Handlungsvorsprungs der Effekte von Naturwissenschaften. ... Die Gesellschaft produziert so ... in sich selbst die Differenz von Technik und Lebenswelt.“ Die gesellschaftliche Ebene der Technik und Naturwissenschaft ist aufgrund ihrer ungebremsten Dynamik der Lebenswelt ihrer geistigen Reflexion immer um einige Schritte voraus. „Daraus erklärt sich eine Ungleichzeitigkeit in beiden gesellschaftlichen Ebenen. Jede betreibt ihre eigensinnigen Eigendynamiken. Die strukturelle scheint dabei die

6 Zitiert nach Friedrich Dessauer, Streit um die Technik, S. 58 f.

7 Gotthard Günthers Technikphilosophie, in: Ernst Kotzmann (Hrsg.), Technologische Kultur, S. 155 – 183.

8 Ebd., S. 163.

weitaus kompetentere zu sein. Sie macht die Vorgaben, der semantischen bleibt – trotz gegenteiligem Selbstverständnis – nur das Reagieren."[9] So kommt es zu den bekannten gegenseitigen Vorwürfen, dass die Techniker und Naturwissenschaftler nicht sehen würden, welchen Zwecken sie dienen und die Geisteswissenschaftler nicht in der Lage wären zu verstehen, was sie kritisieren.

Trotz der Machtstellung der Technik ist der Ingenieur, d.h. der auf wissenschaftlicher Grundlage ausgebildete Techniker, heute mehr denn je heteronom. Er erhält die Spezifikation der zu lösenden technischen Aufgaben in der Regel von anderen und beschränkt sich als technischer Problemlöser darauf, lediglich die Mittel bereitzustellen, die zur Entwicklung der Produkte führen. Darin erschöpft sich seine ganze Kompetenz. Die Frage, welche technischen Entwicklungen verfolgt werden sollen, wird heute fast ausschließlich von wirtschaftlichen Interessen dominiert und damit in anderen Zirkeln entschieden. Das Handeln des Ingenieurs ist nicht nur fremdbestimmt, sondern auch anonym, reproduzierbar, stringent und formal. Er gilt heute als *Homo faber*, der seine Umwelt durch seine technisch-formalen Fertigkeiten und Methoden kontrolliert. Er hat sich dem Mechanismus und der Maschine und damit dem toten Material verschrieben, wobei er selbst in diesem Prozess vollkommen anonym und damit beliebig austauschbar bleibt. Die Vorgehensweisen des Ingenieurs sind gewissermaßen zu formalen Methoden geronnenes öffentliches Denken, das sich des konkreten Intellekts nur bedient. Der Kunst und den Künstlern werden im Vergleich dazu im landläufigen Verständnis entgegen gesetzte Qualitäten zugesprochen. Künstler sind autonom und dem Leben und den Menschen zugewandt, ihr Tun ist kreativ, individuell und singulär. In der künstlerischen Praxis mit computerbasierten Medien ist diese Trennung nicht mehr aufrechtzuerhalten, Wahrnehmung und Kreativität müssen eine neue Verbindung mit stringent rationalem Denken eingehen. Natürlich war die polarisierende Trennung C.P. Snows und seine Rede von den zwei Kulturen schon immer eine Überspitzung, die einer näheren Analyse nie standhielt. Zweifellos gab es immer unter den Literaten, Geisteswissenschaftlern und Künstlern neben den Maschinenstürmern auch eine beträchtliche Zahl von Maschinenanbetern. Umgekehrt gab es schon immer Ingenieure, die ihr Tun im Kontext gesellschaftlicher Verantwortung sahen und auch die Methoden des Ingenieurwesens als allgemeine und fächerübergreifende Entwurfsdisziplin aufgefasst haben. Zum Beispiel sah der bekannte Ingenieur Max Eyth vor 100 Jahren das wesentliche Kennzeichen des technischen Schaffens im schöpferischen Hervorbringen neuer materieller Gebilde. Für ihn bestand der Erfindungsprozess aus drei Phasen, wobei die erste Phase die Konzeption des Gedankens ist, die auf einer nicht rational fassbaren Inspiration beruht. Hierbei handelt der Erfinder ohne äußeren Anlass und wird „von demselben inneren Schaffensdrang und derselben Lust am Schöpfertum getrieben, die auch den

9 Ebd., S. 165.

Künstler beseelen."[10] Viel wichtiger im Hinblick auf die Kritik der zwei Kulturen aber ist, dass das in der breiten Gesellschaft verankerte Bild nicht mehr in der Lage ist, den heutigen Problemen der technologischen Zivilisation gerecht zu werden.

Vor diesem Hintergrund unternimmt das vorliegende Buch den Versuch, einerseits in die Grundstrukturen der Computerprogrammierung unter Java einzuführen und parallel dazu einen Abriss der geistesgeschichtlichen Tradition der programmierbaren Maschine zu leisten. Es soll deutlich werden, dass der Computer zwar eine Technologie des 20. Jahrhunderts ist, seine kulturellen und geistigen Wurzeln aber sehr viel weiter zurückreichen. Angesichts der beschriebenen Kluft zwischen den Kulturen sind wir uns natürlich bewusst, dass die Wirkungen eines solchen Unterfangens nur gering sein werden. Zumal wir zum gegenwärtigen Zeitpunkt nur eine Reihe historischer und technischer Fakten zusammentragen können und bestenfalls erste grobe Konturen des Computers als Medium umreißen. Dennoch sehen wir keine Alternative zur gewählten Vorgehensweise, wenn Computer – und insbesondere Programmierung als Schlüsselzugang zum Computer – nicht länger durch reine Sachbeherrschung geprägt sein sollen, sondern der geistige Sachverstand und die experimentelle Erforschung des Mediums ins Zentrum rücken sollen. Nur weil die primären Strukturen und Methoden der Programmierung den meisten unzugänglich bleiben, sind derart oberflächliche Diskussionen möglich, wie sie seit Jahren in regelmäßigen Abständen durch die Presse gehen. Hier wird Maschinen nicht nur Intelligenz, sondern gar Bewusstsein zugesprochen und bereits das Ende der Menschheit deklariert. Dies aber oftmals auf der schlichten Basis des Vergleichs der Anzahl und Geschwindigkeit menschlicher und maschineller 'Verarbeitungselemente', ohne sich um die eigentlichen Organisationsprinzipien zu kümmern, die dies leisten sollen. Also Sieg durch Quantität und nicht durch Qualität. In diesem Sinne verstehen wir hier Programmierung als technische Reflexionsform und Programme als implementierte Theorien, die in erster Linie bestimmte menschliche Denkformen und menschliches Handeln widerspiegeln. Diese Sedimentationen von Psychostrukturen sind Abstraktionen, die nicht einfach gleichzusetzen sind mit den Strukturen der menschlichen Gehirne, die diese hervorbringen. Gleichzeitig erschließt sich aber durch die algorithmische Synthese und Analyse von Bildern, Klängen und Handlungen zusammen mit den Interfaces als Vermittlungsinstanz zwischen den logischen Strukturen der Maschine und den Wahrnehmungsprozessen ihrer Benutzer ein neuer konzeptueller und experimenteller Raum des Zusammenspiels von Kognition und Wahrnehmung. Natürlich stellen Computer auch heute keine notwendige Voraussetzung dar, um künstlerisch zu arbeiten, noch werden Arbeiten dadurch, dass sie mit Hilfe eines Computers hergestellt werden, a priori besser. Die Geschichte der Kunst des 20. Jahrhunderts macht vielmehr deutlich, dass jedes Material Ausgangsbasis

10 Friedrich Rapp, Analytische Technikphilosophie, S. 11f.

für die künstlerische Praxis und Gegenstand der künstlerischen Reflexion sein kann. Gleichzeitig gilt aber, dass, nachdem die Festlegung auf ein bestimmtes Material erfolgt ist, man sich auf den 'Eigensinn' dieses Materials einlassen muss. Der Begriff des 'Eigensinns'[11] geht davon aus, dass alle Medien etwas je eigenes auszeichnet, dass also auch Computer und Netzwerke je eigene spezifische Potentiale, Strukturen und Beschränkungen zeigen. Diesen Eigensinn gilt es für die Computertechnologien noch herauszuarbeiten, wir stehen dabei am Anfang. Klar ist allerdings, dass der Computer nicht einfach ein weiteres Medium unter Medien ist, sondern sich schon alleine durch seine Fähigkeit auszeichnet, Gedächtnis und intellektuelle Arbeit zu simulieren. Die neuen computerbasierten Medien sehen nur oberflächlich wie herkömmliche Medien aus. Jede Analogie mit Druck, Fotografie, Radio oder Fernsehen bleibt zwangsweise defizitär. Es wird nicht einfach nur die Liste der bekannten Medien um einige weitere Typen verlängert, sondern die organisatorische Basis auf der Medien im Computer realisiert sind, ist eine vollkommen andere. Was an der Oberfläche lediglich wie ein weiterer Medientyp daher kommt, ist in seiner Organisationsstruktur grundlegend verschieden von seinen nicht-programmierten Vorläufern. Dieser wichtige Punkt wird im Laufe des Kapitels noch an verschiedenen Stellen verdeutlicht werden.

1.1 Der Computer als Artefakt

„Was aber ist das für ein Ding, die Maschine, an die sich die menschliche Existenz gebunden hat? Kein Organismus, kein sonstiges Naturobjekt, ein Artefakt. Aber ein Artefakt, das, anders als alle bisherigen Artefakte, sich selbsttätig bewegt. Ein artifizieller Selbstbeweger, also doch wohl eine Entität neuer Art. Und entsprechend entsteht ein neuer Typus von Prozessen. Neben Naturprozesse und Handlungsprozesse treten Maschinenprozesse."
Heinrich Popitz[12]

Im Gegensatz zu natürlichen Gegebenheiten sind technische Systeme künstlich, d.h. vom Menschen hergestellte Artefakte, die im Hinblick auf einen Zweck entwickelt werden. In diesem Zusammenhang wird von Technik häufig auch als 'zweite Natur' gesprochen. Wenn wir uns die erkenntnistheoretischen Vorbedingungen ansehen, ist die Trennung zwischen erster und zweiter Natur aber schwer aufrecht zu erhalten, da wir letztlich beide durch das gleiche logische Auge betrachten. Friedrich Rapp[13] nennt weitere Gründe, warum Technik nicht als Anti- oder Gegennatur verstanden werden kann und Natur und Technik an

11 Vgl. Giaco Schiesser, Media | Art | Education: Arbeit am Eigensinn, in: Gerfried Stocker, Christine Schöpf (Hrsg.), CODE – The Language of our Time, S. 371 – 373.

12 In: Epochen der Technikgeschichte, S. 27.

13 Vgl. Friedrich Rapp, Analytische Technikphilosophie, S. 135 ff.

mindestens drei Stellen miteinander verwoben sind:

1. Technische Systeme lassen sich nur realisieren, wenn sie mit den Naturgesetzen verträglich sind. Technisches Handeln beruht gewissermaßen auf einem strikten Gehorsam gegenüber der Natur. „Der Kunstgriff des technischen Handels besteht gerade darin, daß man – im Gegensatz zum magischen Denken – der Natur keine unrealistischen Ziele aufzwingt, sondern sich möglichst genau ihrem eigenen 'unbeugsamen' Gesetzen anpasst. Innerhalb des dadurch gewonnenen Spielraums hat man dann – unter Berücksichtigung der vorhandenen Ressourcen – völlige Freiheit, um die Naturprozesse im Sinne menschlicher Zwecksetzungen zu lenken." Wir werden sehen, dass der Computer den größtmöglichen Spielraum schafft, indem eine Trennung in Hard- und Software vollzogen wird. Während die Hardware an die physikalischen Naturgesetze gebunden ist, unterliegt die Software nur noch den Grenzen der eigenen Vorstellungskraft.
2. Technik ist verglichen mit den Selbstgestaltungsaspekten der Natur durchaus keine Einzelerscheinung. Technische Erzeugnisse des Menschen sind zum Beispiel durchaus mit den als natürlich eingestuften Eingriffen von Tieren vergleichbar (Biberdämme, Termitenhügel, Vogelnester).
3. Unser Verständnis von Natürlichkeit unterliegt einem historischen Wandel. Oft werden zunächst als unnatürlich geltende Eingriffe und Artefakte nach einer angemessenen Gewöhnungsphase als natürliche Verhältnisse empfunden. Hier verweist Rapp auf Moscovici, der bereits beschreibt, wie zunächst die Arbeit des Bauern als natürlich und die künstliche Arbeit des Handwerkers als unnatürlich empfunden wird. In der weiteren geschichtlichen Entwicklung wird dann aber die Handwerkstätigkeit als selbstverständlich betrachtet und der künstlichen Konstruktionsarbeit des Ingenieurs gegenübergestellt. Rapp weist aber auch darauf hin, dass, obwohl das technische Milieu letzten Endes ebenso natürlich ist wie jedes andere, es für den Menschen Grenzen für das Tempo der Anpassungsfähigkeit an die selbstgeschaffene Umwelt gibt.

Trotz aller Verwobenheit zwischen erster und zweiter Natur sind die Arbeitsprozesse von Naturwissenschaftlern und Ingenieuren verschieden. Im Zentrum der Naturwissenschaften steht das Bemühen, Wissen über natürliche Erscheinungen und Objekte zu erlangen. Im Gegensatz dazu steht die Herstellung künstlicher Objekte mit geplanten Eigenschaften traditionell im Zentrum der Arbeit des Ingenieurs wie auch des Künstlers und Gestalters. Zumindest in Hinsicht auf die Herstellung von Artefakten sind die Ingenieure der Kunst näher als der Naturwissenschaft. Angesichts dieser vollkommen diversen Ausgangslagen – Analyse versus Synthese – hat der amerikanische Nobelpreisträger Herbert Simon[14] bereits in den 1960-er Jahren eine eigenständige Wissenschaft vom Entwerfen gefordert, die sich klar von naturwissenschaftlichen Lehrkonzepten

14 Vgl. Die Wissenschaften vom Künstlichen.

abhebt. Ungeachtet der Schlüsselrolle des Entwurfs in allen anspruchsvollen praktischen Tätigkeiten, haben aber – damals wie heute – die Naturwissenschaften die Wissenschaften vom Künstlichen fast vollständig aus den Lehrplänen verdrängt. Die Notwendigkeit einer Wissenschaft des Artefaktes tritt angesichts der zentralen Rolle des Computers in allen Entwurfs- und Produktionsprozessen nun aber umso deutlicher zutage. Wir müssen damit beginnen, uns eingehender mit den Bedingungen symbolischer Entwurfsprozesse auseinander zu setzen.

1.1.1 Zum Wesen der Technik

„Wie? Das menschliche Leben wäre also in seiner spezifischen Dimension ein Werk der Einbildungskraft? Der Mensch wäre eine Art Dichter seiner selbst, der die phantastische Figur einer Person samt ihrem unwirklichen Typus von Beschäftigungen ersinnt und der, damit ihm diese Erfindung wirklich gelinge, alles tut was er tut, das heißt Techniker ist?"
José Ortega Y Gasset[15]

Da das philosophische Grundthema die 'Vernunft' ist und die Entwicklung des Computers vollständig in der rationalistischen Tradition steht, kann es nicht verwundern, dass es im Verlaufe des Buches immer wieder Bezüge zur Philosophie geben wird. Obwohl es kaum einen Bereich des individuellen oder gesellschaftlichen Lebens gibt, der nicht von Technik durchdrungen und durch Technik geprägt wäre, führt die Behandlung der Technik als philosophisches Problem allerdings bis heute eher ein Randdasein und wird als eigenständige Fragestellung überhaupt erst seit Ende des 19. Jahrhunderts behandelt. Als Begründer der Technikphilosophie[16] gilt gemeinhin der Geographielehrer und Philosoph Ernst Kapp mit seinem 1877 erschienenen Buch *Grundlinien einer Philosophie der Technik*. Da alle technischen Artefakte ihre Existenz der geplanten und zweckorientierten Herstellung durch den Menschen verdanken, konzentrierte sich die philosophische Reflexion der Maschinentechnik zunächst auf die schöpferischen Leistungen des Erfindungs- und Herstellungsprozesses technischer Artefakte. Kapps Arbeit entsteht in einer Atmosphäre, in der die Möglichkeiten der Technik noch allgemein positiv eingeschätzt werden. Er versteht die aus der Hand des Menschen stammenden Artefakte als die eigentliche Bedingung des Menschen zum Selbstbewusstsein und Technik als Materialisierung und Nach-Außen-Setzung innerer Vorstellungswelten, die deshalb anthropologisch gedeutet werden müssen. Für ihn ist die Hand das Vorbild für alle Werkzeuge und so zum Beispiel der Hammer die Nachbildung des Armes mit geballter Faust.[17] Kapps Vorstellungen zur 'Organprojektion' und zur Selbsterlösung des Menschen durch Technik

15 Betrachtungen über die Technik, in: Signale unserer Zeit, S. 467.
16 Vgl. Friedrich Rapp, Analytische Technikphilosophie, S. 11.
17 Ebd.

werden später von Arnold Gehlen und Donald Brinkmann wieder aufgenommen und erweitert. Die Ingenieurssicht Kapps und ähnlicher Arbeiten zur Analyse technischen Handelns wird später auf den Bereich der Kulturphilosophie erweitert und heute in der Soziologie untersucht, die sich vor allem für soziale Probleme der Technisierung interessiert. Es ist nicht möglich, hier auch nur einen groben Überblick über die unterschiedlichen Theorien und Ansätze sowie die inzwischen vielfältige Literatur zum Thema Technikphilosophie zu geben. Einen guten Überblick gibt hier der Band *Nachdenken über Technik*, der vom Verein Deutscher Ingenieure herausgegeben wird.[18] Seit den ersten philosophischen Arbeiten zum Thema Technik hat sich ein grundlegender Wandel vollzogen. Während Kapps Arbeit noch in einer technikaffirmativen Atmosphäre entsteht, überwiegt heute entweder eine generelle Skepsis gegenüber aller Technologie oder es wir eine schizophrene Haltung eingenommen. Einerseits werden neue Technologien ganz selbstverständlich und unreflektiert genutzt und in den Lebensalltag integriert, gleichzeitig wird aber – bei Nachfrage – der Technik kein wichtiger Einfluss auf das eigene Leben oder das kreative Handeln zuerkannt, sie wird im Gegenteil als bedeutungslos oder gar zerstörerisch und unmenschlich deklariert.

Abbildung 1:
Faszination und Schrecken der Maschine. Raoul Hausmann (1886 – 1971), *Mechanischer Kopf – Der Geist unserer Zeit*, Paris 1918, Musée d'Art Moderne. Auf dem Kopf steht ein Becher zum Eintrichtern von Gesetzen und Regeln. Am Hinterkopf ist eine Geldbörse angebracht, die an wirtschaftliche Zwänge erinnert. Statt eines Gehirns ist nur eine rücklaufende Paragraphen-, Dividenden- und Phrasendreschmaschine zu finden.

Nach José Ortega Y Gasset[19] ist das, was wir unser Leben nennen, vor allem der Drang, einen persönlichen Plan zu erfüllen. Demnach ist der Mensch etwas, das weder eine körperliche noch eine geistige Wirklichkeit hat, er ist vielmehr ein Vorhaben, ein Programm. Er ist das, was er noch nicht ist, aber gerne sein möchte. Was der Einzelne Leben nennt, das 'Gut-Leben', 'Sich-wohl-Befinden' sind dabei bewegliche, grenzenlos variable Begriffe, die jede Kultur, jede Generation und darin jeder Einzelne neu für sich prüfen muss. Über die Umwelt, in die wir mit unserem Denken und Tun eingebettet sind, schreibt Ortega Y Gasset: „Was wir Natur, Umwelt oder Welt nennen ist nichts anderes als das reine System der Erleichterungen und der Schwierigkeiten, die der Mensch mit

18 Siehe Literaturverzeichnis Hubig, Huning, Ropohl (Hrsg.).

19 Vgl. Betrachtungen über die Technik, in: Signale unserer Zeit, S. 472.

seinem Programm antrifft.“ An diesem Punkt verankert er das Wesen der Technik. Der Mensch reagiert auf die ihm von der Natur auferlegten Notwendigkeiten, indem er seinerseits der Natur Veränderungen auferlegt, er schafft eine für sich günstigere Umwelt. Doch Technik ist mehr. „Das Tier kann sich nicht aus der Reihe seiner natürlichen Akte, aus der Natur, zurückziehen, weil es nur diese ist und nicht wüsste wohin, wenn es sich aus ihr entfernte. Der Mensch hingegen ist nicht seine Umwelt, sondern er ist nur eingebettet in sie und kann in einigen Augenblicken aus ihr herausspringen, sich in sich selbst versenken, er selbst sein und sich nur mit Dingen beschäftigen, die nicht ein direktes oder unmittelbares Aufmerken auf die Befehle und Notwendigkeiten seiner Umwelt verlangen. In diesen außer- oder übernatürlichen Augenblicken des Selbstseins und Rückzugs in sich selbst erfindet und bewerkstelligt er diese zweite Reihe von Akten: er macht Feuer, baut ein Haus, bestellt ein Feld und montiert ein Automobil.“[20] Technik ist – und das ist im Folgenden wesentlich – nicht darauf beschränkt, die Notwendigkeiten des Lebens erträglicher zu machen. Der Mensch ist nicht primär an den biologischen Notwendigkeiten des Überlebens in einer mit Widerständen aufgeladenen Welt interessiert, z.B. am bloßen ‘Sich-Ernähren’, sondern daran seinen eigenen Lebensplan zu verwirklichen. Der Sinn des Lebens liegt nicht im Lebenserhalt, sondern darin, das Leben mit ‘Überflüssigem’ anzureichern. Im Überflüssigen liegt auch der Ursprung aller Kultur. „Ersparter Aufwand war das Erfolgsrezept unserer Art – genauer gesagt: Menschen haben höheren Aufwand treiben gelernt, um sich gröberen vom Hals zu halten. Der Sammelbegriff für diese Aktivität lautet ‘Kultur’. Freilich, sie erschöpft sich nicht in bloßer Ökonomie. Kultur in vollem Wortsinn gibt es nicht ohne Überfluss; und vielleicht fängt sie erst da recht an, wo dieser Überfluss nötig, notwendig, lebenswichtig gefunden wird.“[21] Um diese Freiräume jenseits der auferlegten Notwendigkeiten zu ermöglichen, ist Technik erforderlich. Die Probleme des Menschen können somit nur technisch gelöst werden! Wenn wir allen technischen Ballast abwerfen, kommt nicht etwa ein befreites Wesen – der Mensch an sich – zum Vorschein, sondern es bleibt wenig, dem wir die Qualität ‘Mensch’ zuschreiben würden. Wenn es beispielsweise darum geht, Knochenfunde historisch einzuordnen und zu entscheiden, ob es sich um menschliche oder tierische Überreste handelt, gelten Werkzeuge als untrügliches Zeichen des Menschseins. „Die Werkzeugtechnik ist offenkundig ein Kennzeichen des Lebewesens Mensch. ... Erst der Sinanthropus macht es aufgrund der mitgeführten Steinwerkzeuge unzweifelhaft, dass Wesen mit dieser Schädelform als Menschen anzusprechen sind. Die Schädelform – mit ihren Hinweisen auf Umfang und Struktur des Gehirns – genügt allein nicht.“[22] Ein vehementer Gegner dieser Sichtweise ist Lewis Mumford. Er bezweifelt die führende Rolle der Werkzeuge

20 Ortega Y Gasset, a.a.O., S. 453.

21 Adolf Muschg, Wer nicht spielen kann, lernt nicht, in: FAZ – Ereignisse und Gestalten Nr. 167, 20.07.1996.

22 Heinrich Popitz, Epochen der Technikgeschichte, S. 40.

bei der Entwicklung des Menschen.[23] „Doch die Definition des Menschen als werkzeugherstellendes Tier ist mittlerweile so fest verwurzelt, dass der bloße Fund von Fragmenten kleiner Primatenschädel neben abgeschlagenen Kieseln, wie dies bei den Australopithecinen in Afrika der Fall war, ihrem Entdecker, Dr. L. S. B. Leakey, ausreichend erschien, um in seinem Fund trotz bedeutender physischer Divergenzen gegenüber den Menschenaffen wie auch den späteren Menschen einen direkten Vorfahren des Homo sapiens zu sehen. Da Leakeys Subhomoniden nur ungefähr ein Drittel der Gehirnkapazität des Homo sapiens hatten – sogar weniger als manche Menschenaffen –, erforderte offenbar die Fähigkeit, grobe Steinwerkzeuge herzustellen und zu verwenden, nicht die reiche zerebrale Ausstattung des Menschen, noch bewirkte sie diese." Mumford bezieht sich in seinen Argumentationen gegen die Dominanz der Werkzeuge bei der Entwicklung des Menschen auch auf den holländischen Historiker Johan Huizinga und seine Arbeit *Homo Ludens – Vom Ursprung der Kultur im Spiel*. Huizinga vertritt dort die These, dass das Spiel viel mehr als die Arbeit das formative Element in der menschlichen Kultur sein muss. Erst Ritual, Nachahmung, Sport, Spiel und Theater befreien den Menschen von seiner Verwurzelung im Tierreich. Laut Mumford war die These, dass der Mensch aber weder Homo sapiens noch Homo ludens, sondern Homo faber sei, bereits so stark im westlichen Denken verankert, dass die Archäologen des 19. Jahrhunderts die paläolithischen Höhlenmalereien, die 1879 in Altamira in Spanien entdeckt wurden, zunächst als Fälschungen deklariert wurden.

Aus der Feststellung, dass der Mensch nicht in erster Linie eine körperliche oder geistige Existenz, sondern ein Vorhaben ist, das sich selbst definieren und verwirklichen muss, folgt aber auch unmittelbar, dass Technik sekundär ist, ihr müssen immer Lebenspläne vorausgehen. Die größte Gefahr für den Menschen sind somit nicht die Hindernisse und Notwendigkeiten, die uns die Umwelt oder gar die Technik auferlegen, sondern enggeführtes Denken, das nicht mehr zu einem eigenen qualitativen Lebensentwurf fähig ist. Natürlich ist der Begriff des freien 'Daseinsentwurfs' hoffnungslos humanistisch. So wenig, wie es ein vorsprachlich gegebenes Erkenntnissubjekt gibt, können wir uns in unserem Lebensentwurf der Einbettung in die bereits gegebene technische Lebenswelt entziehen. Auch wenn Technik und technische Akte – so wie Ortega Y Gasset es sich vorstellt – keine bewusst gesteuerten mittelbaren oder unmittelbaren Funktionen bei der Verwirklichung des primären Plans des Menschseins erfüllen können, greift sie dennoch entscheidend in jede persönliche Lebensführung ein. Wir können mindestens drei Bereiche technischen Handels unterscheiden: (1) Technik als Anstrengung, Anstrengungen zu vermeiden, (2) Technik als Mittel zur Erzeugung des Überflüssigen, und (3) Technik als zweckfreies Handeln und gewählter Widerstand.

23 Lewis Mumford, Mythos der Maschine, S. 16.

Ad (1) Technik als Anstrengung, Anstrengungen zu vermeiden

Die erste Mission der Technik ist für José Ortega Y Gasset, dem Menschen die Freiheit zu geben, er selbst sein zu können. Er charakterisiert Technik als eine Anstrengung, Anstrengungen zu ersparen. Technik ist die Strategie des Menschen, alle unangenehmen Tätigkeiten zu vermeiden, die uns unsere Umwelt aufnötigt. Eine geringere Anstrengung soll eine weitaus größere vermeiden. Die Notwendigkeiten des biologischen Existierens werden dabei nicht unmittelbar befriedigt, sondern die Anstrengung kommt erst der technischen Erfindung zugute, die dann zur Ausführung eines Plans verwendet wird. Dass Technik diese Funktion der Erledigung lästiger Arbeit, sei es körperliche oder geistige, überhaupt erfüllt, wird heute mehr denn je bezweifelt. Unsere Arbeits- und Lebenswelt ist inzwischen tiefgreifend und umfassend technisch geprägt und ausdifferenziert. Scheinbar einfache tägliche Handlungen und Arbeitsvorgänge sind in komplexe technische und organisatorische Zusammenhänge eingebettet, die vom Einzelnen in ihrer Gesamtheit nicht mehr zu erfassen sind. Problemlösungen nehmen so zwangsweise lokalen Charakter an, Fern- und Nebenwirkungen sind nicht mehr zu überschauen. Viele Probleme, die Technik vorgibt zu lösen, werden so überhaupt erst durch sie geschaffen. Die rationalistische Weltsicht stößt hier an ihre Grenzen. Unproduktiver Umgang mit Technik hat wesentlich mit grundlegenden Missverständnissen zu tun, wie technische Artefakte in menschliches Handeln eingebettet sind und was sie für die Gesellschaft und den Einzelnen leisten können. Dem Gefühl des Ausgeliefertseins in einer technischen Welt stehen anderseits noch immer weit verbreitete Wahnvorstellungen von grenzenloser technischer Machbarkeit gegenüber. Unreflektiertes Fortschrittsdenken versucht die Entstehung des Neuen nicht als risikobehafteten Wandel und instabilen Umformungsprozess zu verstehen, sondern geht von stetigem kulturellen Aufstieg aus und zieht die Möglichkeit einer drastischer Komplexitätsreduktion oder des kulturellen Rückschritts überhaupt nicht mehr ins Kalkül. Es wird nicht erkannt, dass unsere verfügbaren rationalen Modelle nicht mehr in der Lage sind, die entstandenen Komplexitäten gezielt zu steuern. Modelle, die von prinzipieller Unkontrollierbarkeit vielschichtiger komplexer (sozialer, politischer, technischer, etc.) Systeme ausgehen, fehlen. Die durch Technik vermittelte Scheinsicherheit verdeckt offensichtlich zu leicht, dass ein wesentliches Merkmal des menschlichen Lebens, auch des vermeintlich geplanten, das Wagnis ist. Festzuhalten ist dennoch: Mittels Technik können existentielle Notwendigkeiten, z.B. des Sich-Ernährens oder des Schutzes vor Kälte, durch die Arbeit einiger Weniger sichergestellt werden. Insofern ermöglicht Technik das Überflüssige. In technologischen Zivilisationen widmet sich nur ein geringer Prozentsatz der Bevölkerung der Herstellung der primären Güter.

Ad (2) Technik als Mittel zur Erzeugung des Überflüssigen

Wenn Technik eine Anstrengung ist, um eine weitaus größere zu vermeiden, dann ist für José Ortega Y Gasset die Wendung dieser Erkenntnis

noch interessanter, nämlich die Frage: Wohin soll die ersparte Anstrengung führen? Was soll mit der gewonnen Zeit, der gesparten Kraft und den offen Möglichkeiten angestellt werden? Welche Tätigkeiten sollen das Leben ausfüllen? Diese neuen Tätigkeiten sind dem Menschen nicht mehr von der Natur auferlegt, er muss sie selbst erfinden. Technik schafft Freiräume und Horizonte, die für die Erzeugung des Überflüssigen genutzt werden können. Das Überflüssige ist aber nach José Ortega Y Gasset das einzig Notwendige, nicht für das bloße tierische Existieren, aber für das Menschsein. Das menschliche Leben ist damit immer schon ein Werk der Einbildungskraft und Technik ist in seiner zweiten Funktion ein Weg, um die natürliche Wirklichkeit zu überschreiten und das zu erfindende Leben zu realisieren. Ein anderer Weg, die gewonnene Zeit zu verbringen, wäre die Meditation, das physische Nicht-Handeln. Welche Bedeutung und welchen Reichtum das Überflüssige entwickeln kann und welche Rolle die Technik dabei einnimmt, kann am Beispiel der Musik verdeutlicht werden. Die Entstehung und Ausdifferenzierung der Musik ist nur in Verbindung mit der Entwicklung der zugehörigen handwerklichen Technik, also der Musikinstrumente und deren Spieltechniken zu verstehen. Als natürliches Instrument steht uns nur die Stimme zur Verfügung, alle anderen Instrumente sind technische Artefakte, die erfunden und hergestellt werden. Dabei genügt es nicht, mit einem herumliegenden Stock gegen einen Baumstumpf zu schlagen, um von Werkzeugherstellung oder Technik sprechen zu können. Das Wesen jedes Werkzeugs und des menschlichen Handels allgemein ist ein anderes: „Nicht den zufälligen Gebrauch eines vor Augen liegenden Werkzeugs für nächste Zwecke, sondern die Herstellung eines Werkzeugs für einen fernen Zweck halten wir für menschliches Tun.“[24] Im Falle der Musik produzieren wir mit Hilfe von Instrumenten Klänge – heute würde man virtuelle Realitäten sagen – die in der Natur so nicht vorkommen. Der Reichtum und die Vielfalt der Musik setzt den Instrumentenbau voraus und umgekehrt. Die Entwicklung der Musik ist nur als gemeinsame Bewegung von Instrumentenbau, Musik-Denken und musikalischer Praxis zu verstehen. Auch bei der Verbreitung und der kulturellen Vererbung der Musik nimmt die Technik eine zentrale Rolle ein. Papier und Schreibwerkzeug sind technische Artefakte, Notennotationssysteme eine Kulturtechnik. Wie alle technischen Artefakte sind auch Musikinstrumente halb Natur und halb Geist.[25] Sie sind Natur insofern, als sie ein sicht- und greifbares, d.h. materielles Stück der Außenwelt sind, und Geist insoweit, als die Natur von selbst keine technischen Artefakte hervorbringt, sie sind vom menschlichen Geist erfunden. Menschliches Denken ist in der Lage, der materiellen Außenwelt durch bewusste Formung intendiertes funktionales Verhalten aufzuzwingen. Die instabilen technischen Artefakte haben aber die Tendenz, sich von beiden Seiten, der Natur wie dem Geist, abzulösen und eine eigenständige dritte Existenz zu bilden. Die von uns erfundenen Objekte schlagen zurück auf unser Bewusstsein,

24 Arnold Gehlen, Der Mensch – Seine Natur und seine Stellung in der Welt, S. 68.

25 Vgl. Gotthard Günther, Das Bewusstsein der Maschinen, S. 181.

wir begegnen Ihnen als Realisation in der Außenwelt neu und fremdartig. Wenn wir über Technik reden, so meist als einer von uns unabhängigen, eigenständigen äußeren Existenz. Hollywoodsche Zukunftsphantasien, in denen Maschinen die Kontrolle über die Welt an sich reißen, haben hier ihre Wurzeln. Verstärkt wird diese Neigung dadurch, dass wir in der Regel die technischen Gegenstände, denen wir im Alltag begegnen, nicht mehr selbst herstellen. Wir wissen von den wenigsten Artefakten, wie ihr Innenleben, d.h. die technische Struktur aussieht.

Ad (3) Technik als zweckfreies Handeln und gewählter Widerstand

Die ersten beiden Funktionen der Technik können dank eines organisatorischen Tricks – unserer Gesellschaftsform der Spezialisierung – bequem den Technikern und Ingenieuren überlassen werden. Es reicht aus, wenn die Funktionen der technischen Artefakte allen zur Verfügung gestellt werden. Eine der wichtigsten Freiheiten, die wir durch Technik erlangen können, ist Nutzen daraus zu ziehen, ohne uns wirklich mit Technik beschäftigen zu müssen. Erreicht wird dies durch die Trennung von Funktion und innerer Organisation. So ist zum Beispiel der Film ohne Technik nicht denkbar, dennoch kann der Filmemacher vollkommen untechnisch bleiben. Kaum ein Musiker kann selbst ein Instrument bauen, kaum eine Autofahrer versteht die Technologie eines modernen Fahrzeuges. Wohlstand bedeutet gerade, dass man nicht mehr selbst alles herstellen muss, sondern sein Tun auf Weniges, Selbstgewähltes beschränken kann. Hoher gesellschaftlicher Status gilt heute gerade dann als erreicht, wenn man nicht genötigt ist, sich an den Herstellungsprozessen der technischen Alltagsgegenstände zu beteiligen, sondern sich der Produktion von Kulturgütern widmen kann. Der Computer als kulturelles Produkt war – wie viele andere Technologien – von vorne herein nicht an die Befriedigung von Elementarbedürfnissen geknüpft. Er wurde ursprünglich vielmehr verstanden als Hilfsmittel zur Befreiung des Menschen von Routineaufgaben des Geistes. Insofern diente er zwar dem oben beschriebenen Zweck, Anstrengungen zu vermeiden, wenn wir uns heutige typische Computeranwendungen ansehen, stellen wir fest, dass der Computer auch zunehmend der Erzeugung des Überflüssigen im Sinne Gassets dient. Aber hier endet gleichzeitig die Analogie zu Ortega Y Gasset. Eine anscheinend nicht ganz so offensichtliche Möglichkeit der Technik ist die dritte Funktion – Technik als freies Handeln und selbstgewählter Widerstand. Diese Anwendungsform gehört in gewisser Weise zum Bereich des Überflüssigen, andererseits stehen hierbei nicht die Bequemlichkeiten und Genüsse die technische Artefakte bieten im Zentrum, sondern die Auseinandersetzung mit ihrer Struktur. Nachdem der Mensch die Anstrengungen, welche die Natur ihm auferlegt, überwunden hat, erfindet er diese neu. Diese neuen Mühen sind nun aber selbst gewählte Anstrengungen, denen er sich unter selbst bestimmten Bedingungen freiwillig unterzieht. Sport kann als Beispiel angeführt werden, wie ursprünglich existentielle körperliche Anstrengungen kulturell überformt werden. Wir hatten oben bereits auf Huizinga verwiesen, der die These vertritt, dass das Spiel viel mehr als die

Arbeit das formative Element in der menschlichen Kultur sei und, dass erst Ritual, Nachahmung, Sport, Spiel und Theater den Menschen von seiner Verwurzelung im Tierreich befreien. Lange bevor der Mensch die Fähigkeiten erworben hatte, seine Umwelt aktiv umzugestalten, hatte er sich symbolische Ersatzwelten geschaffen, in denen er seine menschlichen Interessen nach belieben Formen und Gestalten konnte. In dieser Hinsicht sind Computer und Programmstrukturen interessante Werkzeuge, um symbolische Ersatzwelten zu erzeugen und sich mit diesen auseinander zu setzen. Darüber hinaus ist Programmierung eine spezielle Form der Auseinandersetzung mit den eigenen Denkstrukturen. Nicht das Endprodukt ist hier das Entscheidende, sondern die Arbeit mit dem Material und die Überwindung von Widerständen.

1.1.2 Programme als Realitätskonstruktionen

„Any program is a model of a model within a theory of a model of an abstraction of a portion of the world or of some universe of discourse."
Meir M. Lehman[26]

Wie ist es zu erklären, dass die Ergebnisse der modernen Naturwissenschaften – und nicht nur diese – durchweg programmierbar sind? Natur- und ingenieurswissenschaftliche Forschung wird heute auf allen Gebieten durch die programmierbare Maschine hindurch betrieben. Der Computer ist das universelle Werkzeug, das an der Produktion neuer Erkenntnisse maßgeblich beteiligt ist, etwa bei der Erfindung neuer Werkstoffe, bei komplexen Beweisführungen in der Mathematik oder bei der Simulation von Sternsystemen. Nicht viel anders ist die Situation in den Entwurfsdisziplinen, etwa dem Design, der Architektur oder den vielfältigen Ingenieurswissenschaften. Wie ist es möglich, dass ein und dasselbe Artefakt für so unterschiedliche Disziplinen ein zentrales Werkzeug ist? Entscheidend hierfür ist, dass der Computer keine Maschine ist, die klassisch mechanischen Prinzipien folgt, sondern eine transklassische Maschine. „Die Arbeitsweise der klassischen Maschine folgt dem Vorbild des arbeitenden Armes (samt Hand). Die Idee der transklassischen Maschine aber erwächst aus den technischen Forderungen, einen Mechanismus zu entwickeln, der nach der Analogie des menschlichen Gehirns arbeitet."[27] Auch wenn die Idee der 'künstlichen Intelligenz' (KI) wahrscheinlich aus dieser Analogie erwachsen ist, geht es bei dieser Aussage nicht primär um den Vergleich mit der KI. Vielmehr wird die These vertreten, dass Computer als materialisierte Projektionen von Wesensmerkmalen des Menschen begriffen werden können. „Technologie ist nichts anderes als die gegenständlich gewordene Widerspiegelung der menschlichen Seele in die Natur. Maschinen sind vom Menschen produziert.

26 In: Programs, Life Cycles, and Laws of Software Evolution, S. 1061.

27 Vgl. Gotthard Günther, Das Bewusstsein der Maschinen, S. 184.

Sie sind nichts anderes als die Materialisierung dessen, was im Kopf, in der Psyche des Menschen bereits vorhanden ist. Maschinen können als materialisierte Projektionen von Wesensmerkmalen des Menschen begriffen werden. Nicht die Technik wäre dann das größte Problem des gegenwärtigen Menschen, sondern der Mensch selbst."[28] KI betrachtet lediglich ein ganz bestimmtes zielgerichtetes Problemlösungsverhalten und ist damit nur ein kleiner Ausschnitt aus der sich in jedem Algorithmus entäußernden Psychostruktur und dem Denken des Menschen. In dieser Hinsicht ist der Computer ein Schlüssel zu unserem Selbstverständnis, eine evokatorische Maschine, die uns zum Nachdenken über uns selbst zwingt. Es stellt sich natürlich die Frage, wie es überhaupt möglich ist, dass ein Denkprozess von einer Maschine ausgeführt werden kann? Die Antwort ist einfach: Der Denkprozess muss in einen Handlungsprozess übersetzt werden. Hierbei kommt der Formalisierung eine entscheidende Rolle zu, d.h. der sprachlich eindeutigen Erfassung von Methoden. Im zweiten Abschnitt dieses Kapitels wird dies ausführlicher erläutert werden. Methoden sind in diesem Verständnis Regeln für unser Handeln, die es gestatten, aus unendlich vielen Handlungsalternativen eine bestimmte auszuwählen. Bereits geleistete Verständnisarbeit wird mit Hilfe formaler Strukturen funktionalisiert und damit zu erlernbarem Erkenntnisbesitz und zum übertragbaren Verfahren. „Formalisierung ist nichts anderes als die handlichste Art solcher Funktionalisierung des einmal Geleisteten; aber sie ist eben auch potentiell schon Technisierung, denn was formalisiert werden kann – das heißt: was seine Anwendbarkeit unabhängig von der Einsichtigkeit des Vollzuges gewinnt –, das ist im Grunde auch schon mechanisiert, auch wenn die realen Mechanismen zu seiner Speicherung und geregelten Assoziation nicht bereit gestanden haben. Alle Methodik will unreflektierte Wiederholbarkeit schaffen, ein wachsendes Fundament von Voraussetzungen, das zwar immer mit im Spiele ist, aber nicht immer aktualisiert werden muss."[29]

Die Organisationsprinzipien, die wir in der programmierbaren Maschine finden und die wir später noch eingehender betrachten werden, sind die Prinzipien des 'mechanistischen Denkens' und einer 'mechanistischen Weltauffassung'. Mechanisierung ist quasi das Endergebnis einer rationalen Einstellung zur Welt. Die Formalisierung, d.h. die zeichenbasierte Beschreibung von Methoden, setzt Abstraktion voraus. Im Vorgang der Abstraktion findet eine Verallgemeinerung und Ablösung von den konkreten Phänomenen statt. Die Abstraktion eines Produktionsvorganges beruht auf der gedanklichen Zerlegung eines Vorgangs in seine Teilprozesse. Nach dieser Zerlegung muss eine erneute Zusammensetzung erfolgen, die nun die Relationen und das Zusammenwirken der Teilprozesse berücksichtigt. Abstraktion und Formalisierung sind damit immer Analyse und Synthese zugleich. Mechanisierung ist das Wissen über die Umsetzbarkeit des

28 Arno Bammé et. Al., Maschinen-Menschen Mensch-Maschinen – Grundrisse einer sozialen Beziehung, S. 110.

29 Hans Blumenberg, a.a.O., S. 41 f.

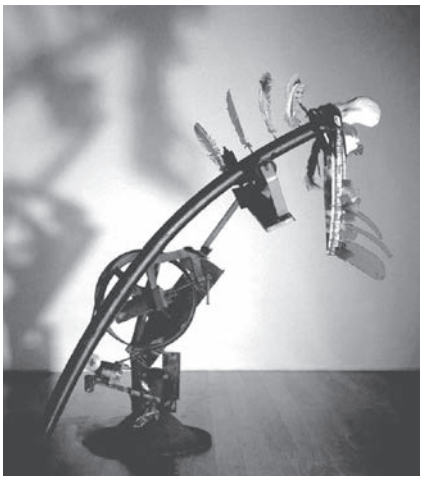


Abbildung 2/3:
Jean Tinguely, *Jacob Burckhardt* (links), *Jean-Jacques Rousseau* (rechts). Ausgerechnet Rousseau, der kategorisch ablehnt, dass Maschinen denken können, findet in den Philosophenmaschinen Tinguely's seine Reinkarnation als Maschine.

Formalen in prozessierende Materie. Das Zusammenwirken der drei Prinzipien – Abstraktion, Formalisierung und Mechanisierung – bilden zusammen den eigentlichen Kern der mechanistischen Weltauffassung. Diese Grundprinzipien des mechanistischen Denkens sind nach Sohn-Rethel aber nicht a priori gegeben, sondern haben ihren Ursprung selbst wieder im gesellschaftlichen Sein. Letztlich gründet die abstrakte Auffassung von der Natur – so seine Theorie – in einer Art Urabstraktion, nämlich der Abstraktion des Warentausches.

„[Die abstrakte Natur] ist, um es paradox zu beschreiben, die Natur, die übrigbleibt, wenn die konkrete Natur, nämlich die im Gebrauch der Waren engagierte, die Stofflichkeit der Dinge verändernde Natur, abgezogen worden ist. Daher ihr Charakter eines absoluten, realen und doch unsinnlichen Minimums von Naturvorgang. Diese abstrakte Natur beschreibt sich in reinen, alles Wahrgenommene, qualitativ Sinnliche der Quantifizierung unterwerfenden Begriffen und Prinzipien, die nirgends ihren Sinn haben und verstanden werden können als unter 'zivilisierten Menschen', Menschen nämlich, die wissen, was Warentausch und was Geld ist und die darin unterstellten Eigentumsbegriffe teilen, kurz Menschen, die in einer durch und durch synthetischen Gesellschaft leben. Wir können diese Begriffe und Prinzipien unter dem Namen 'mechanistisches Denken' zusammenfassen. Es sind Begriffe und Prinzipien, die nur im menschlichen Denken existieren, – aber nicht aus dem Denken entspringen. Sie sind Begriffe und Grundsätze des 'reinen Verstandes' im Sinne Kants."[30]

Für unseren Zusammenhang ist nicht wichtig, wo die Auffassung von der

30 Alfred Sohn-Rethel, Geistige und körperliche Arbeit, S. 89.

‘objektiven Realität’ und die mit der Abstraktion angestrebte Allgemeingültigkeit ihren Ursprung hat, ob im Subjekt (Kant), in der Außenwelt (Simon) oder im gesellschaftlichen Sein (Sohn-Rethel). Wichtig ist, dass Abstraktion, Formalisierung und Mechanisierung eng miteinander verbunden sind und zusammen die Voraussetzung für die Programmierung bilden. In der Programmierung müssen wir uns um die Mechanisierung im engeren Sinn, d.h. die Umsetzung der formalen Beschreibung in prozessierende Materie, nicht mehr kümmern. Dies erledigt der Compiler zusammen mit der universellen Hardware für uns. Abstraktion und Formalisierung sind dagegen Kernaufgaben jedes Softwareentwicklungsprozesses. Im Allgemeinen wird dieser Nukleus der Softwareentwicklung als ‘Modellierung’ bezeichnet. Wenn wir verstehen wollen, was Programmierung bedeutet und in welchem Sinne es sich hier um Realitätsentwürfe handelt, müssen wir uns mit dem Modellbegriff auseinander setzen.

Herbert Stachowiak beschreibt drei Hauptmerkmale des allgemeinen Modellbegriffs.[31] 1. Das Abbildungsmerkmal, 2. das Verkürzungsmerkmal, 3. das pragmatische Merkmal. Das Abbildungsmerkmal besagt, dass Modelle stets Modelle von etwas sind. Modelle sind demnach immer Abbildungen natürlicher oder künstlicher Originale, die selbst wieder Modelle sein können. Der Begriff des Originals ist dabei denkbar weit gefasst. „Jede von einem natürlichen oder maschinellen kognitiven Subjekt erfahrbare, allgemeiner: ‘erstellbare’ Entität kann in diesem umfassenden Sinn als Original eines oder mehrerer Modelle aufgefasst werden.“[32] Das Verkürzungsmerkmal bringt zum Ausdruck, dass Modelle im Allgemeinen nicht alle Attribute eines Originals erfassen, sondern nur solche, die den jeweiligen Modellschaffern und/oder Modellnutzern relevant erscheinen. Das pragmatische Merkmal schließlich macht deutlich, dass Modelle ihren Originalen nicht per se eindeutig zugeordnet sind, sondern nur für bestimmte Modellnutzer, innerhalb bestimmter Zeitgrenzen und nur für bestimmte Operationen. „Über die abbildungsmäßige Originalbezogenheit hinaus ist mithin der allgemeine Modellbegriff dreifach pragmatisch zu relativieren. Modelle sind nicht nur Modelle von etwas. Sie sind auch Modelle für jemanden, einen Menschen oder einen künstlichen Modellbenutzer. Sie erfüllen dabei ihre Funktionen in der Zeit, innerhalb eines Zeitintervalls. Und sie sind schließlich Modelle zu einem bestimmten Zweck. Man könnte diesen Sachverhalt auch so ausdrücken: Eine pragmatisch vollständige Bestimmung des Modellbegriffs hat nicht nur die Frage zu berücksichtigen, wovon etwas Modell ist, sondern auch, für wen, wann und wozu bezüglich seiner je spezifischen Funktionen es Modell ist.“[33] Obwohl Programme formale Gebilde sind, die technisch realisiert werden, unterliegt der Modellbildungsprozess subjektiven Perspektiven der Beteiligten und muss gleichsam als sozialer Vorgang der Entwicklung und Aneignung verstanden

31 Vgl. Allgemeine Modelltheorie, S. 131 f.

32 Ebd.

33 Ebd., S.133.

werden.[34] Vereinfacht ausgedrückt: Wäre das Programmieren ein stringenter Prozess, der nach festen Regeln abläuft, dann wäre es längst automatisiert worden. In der Softwareentwicklung können wir drei verschiedene Ebenen der Modellbildung unterscheiden:[35] 1. Das Anwendungsmodell, das eine Repräsentation des Gegenstandsbereiches anstrebt. Von Modellierung kann gesprochen werden, wenn es gewisse Analogien zwischen der Struktur des Modells und der Struktur des Referenzbereiches gibt, der modelliert werden soll. 2. Das formale Modell – oder die Spezifikation – stellt eine Verbindung zwischen formal-logischen Handlungen und ihrer Bedeutung her. 3. Das Berechnungsmodell schließlich ist das Programm selbst, d.h. eine Einheit von Handlungsanweisungen, die in einer bestimmten Programmiersprache formuliert und auf mit entsprechendem Interpreter (Compiler) ausgestatteten Rechenmaschinen ablauffähig sind. Im künstlerischen Kontext wird das Anwendungsmodell oft nicht detailliert ausgeführt und die Spezifikationsebene wird ganz weggelassen, vielmehr sind Programm und Anwendungsmodell im Prozess der Programmierung viel enger miteinander verwoben.

Die schwierige Aufgabe der Modellbildung kann natürlich nicht losgelöst von der Komplexität des Anwendungsbereichs (auch Gegenstandsbereich, Referenzbereich) gesehen werden. Lehman[36] unterscheidet drei verschiedene Softwareklassen, die er S, P und E Kategorien nennt, und in die sich die unterschiedlichen Anwendungen mit ihren spezifischen Modellierungsanforderungen einordnen lassen. S-Programme sind Programme, die sich vollständig von ihrer Spezifikation ableiten lassen. Typische Beispiele sind mathematische Probleme wie die Suche nach dem kleinsten gemeinsamen Teiler zweier Zahlen, die Auswertung einer Funktion an bestimmten Stellen, oder die Erzeugung eines Kreises an einer genau beschriebenen Stelle auf dem Bildschirm. Zwar können diese Aufgaben sich auch auf Bereiche der externen Welt (den Anwendungsbereich) beziehen, diese Beziehung ist aber nicht kausal, sondern flüchtig und interpretationsabhängig und für die Lösung des Problems auch nicht notwendig. Diese S-Probleme zeichnen sich dadurch aus, dass die Lösungsprogramme sich exakt definieren lassen. Bei P-Programmen dagegen kann der gesamte Problembereich nicht mehr so präzise eingegrenzt werden. Beispiele hierfür sind der schachspielende Computer oder die Wettervorhersage. Obwohl die erlaubten Aktionen und zugrunde liegenden Regeln vollständig bekannt sind und auch das Problem exakt beschrieben werden kann, sind nur approximative Lösungen errechenbar. Die Beschreibung des Problembereichs ist hier enger verbunden mit Situationen in der Realwelt. Es handelt sich dabei um Modelle von Abstraktionen von Situationen der Realwelt. Die Modellbildung beinhaltet hier in der Regel Ungewissheiten, unbekannte Einflüsse, willkürliche Annahmen

34 Vgl. Cristiane Floyd, Ralf Klischewski, Modellierung – ein Handgriff zur Wirklichkeit.

35 Ebd., S. 22.

36 A.a.O., S. 1061 ff.

und kontinuierliche Variablen. Während in S-Programmen die Bewertung einer Lösung ausschließlich über die Spezifikation der Aufgabe erfolgen kann, kann die Qualität von P-Programmen nur eingeschätzt werden, indem man die Leistung des Programms in der Realwelt begutachtet. Signifikante Abweichungen zwischen den in der Realwelt beobachteten und den errechneten Werten führen zu Änderungen der Einschätzung des Problems. Die gemessenen Differenzen können an allen Stellen des Softwareprozesses zu Änderungen führen: in der Wahrnehmung des Problems, seiner informellen Beschreibung, der Modellbildung, bis hin zur Implementierung. P-Programme verlieren deshalb entweder im Laufe der Zeit an Effektivität oder sie müssen ständig an die sich ändernden Bedingungen angepasst werden. Ganz allgemein gehören deshalb alle Simulationssysteme zur Klasse der P-Programme. S- und P-Programmen ist gemeinsam, dass der Gegenstandsbereich abgeschlossen und überschaubar ist, trotz unsicherer Modellannahmen und der Verbindung des Programms mit der unscharfen Realwelt. Die dritte Kategorie von Programmen unterliegt einem noch größeren Druck, sich an ändernde Verhältnisse anzupassen. E-Programme sind Systeme, die Teil des Anwendungsbereiches werden, den sie Modellieren, d.h. sie sind eingebettet in die Realwelt und verändern diese durch ihre eigene Aktivität. Die Programme werden in der Realität wirksam, d.h. es gibt in diesen Systemen eine elementare Rückkopplungsschleife indem die Systeme das was sie modellieren auch modifizieren. Die Modifikation der Realität muss dabei nicht durch direkte Handlungen erfolgen, wie etwa in der Robotik, sondern sie wird in der Regel indirekt wirksam sein, indem sie die Sichtweisen und Arbeitsprozesse der Benutzer verändert. Natürlich sind auch bei eingebetteten Systemen die Eigenschaften und das Verhalten des Systems vollständig durch die Programmanweisungen determiniert. Die Wechselwirkungen zwischen Programm und Umgebung können allerdings sehr komplex sein.

Jeder Entwickler ist gut beraten, sich – zusammen mit obigen Hauptmerkmalen des Modellbegriffs von Stachowiak – folgende metatheoretischen Grundannahmen der Modellierungspraxis zu vergegenwärtigen:[37] 1. Modelle sind nicht als Realitätsabbilder zu verstehen, sondern sie rekonstruieren im wesentlichen die Begriffe und die Sprache, mit der im Anwendungskontext Sinnzusammenhänge beschrieben und kommuniziert werden. Modelliert wird nicht eine objektive Welt, sondern die Sicht der beteiligten Akteure. 2. Modelle vergegenständlichen Abstraktionen und wirken damit normativ. Durch diese Integrationsfunktion konstruieren sie Realitäten. Modelle tragen damit zur Aufrechterhaltung einer sozial konstruierten Realität bei und kontrollieren Handlungen, indem sie die Wahrnehmung, Begründungszusammenhänge, usw. kanalisieren. 3. Modelle sind abhängig von Vorurteilen und dem Vorverständnis, mit dem die Modelle entwickelt werden. Diese Vorurteile lassen sich nur durch einen reflektierten Dialog transparent machen. 4. Die Modellierung verändert den Bezugsrahmen

37 Vgl. Floyd, Klischewski, a.a.O., S. 23 f.

für die Handlungsorientierung und greift damit in die Interessen der beteiligten Akteure ein. Formale Modelle sind keine isolierten Gebilde, sondern werden in sozialen und kulturellen Prozessen konstituiert und anschießend in diesen Feldern wirksam. Als selbst ablaufende Modelle haben sie direkte Auswirkungen auf die Realität und verändern die Bedingungen sozialen und kulturellen Handelns.

Bei Entwurfsprozessen – oder allgemein in der künstlerischen und medialen Verwendung des Computers – haben wir es aber immer mit sozial und kulturell eingebetteten Systemen zu tun, deren Gegenstandsbereich nur subjektiv abgeschlossen werden kann. Im Prozess der Modellierung findet eine sozial komplexe, gleichzeitig subjektive und vorurteilsgesteuerte teilweise willkürliche Abgrenzung und Determinierung des Gegenstandsbereichs statt. Entwurfsprozesse werden üblicherweise allerdings nicht in S-, P- und E-Klassen unterteilt, sondern in wohl definierte und schlecht definierte Probleme. Wohl definierte Probleme zeichnen sich dadurch aus, dass jeweils eine systematische Methode zur Verfügung steht, ob eine akzeptable Lösung gefunden wurde. Darüber hinaus sind sowohl alle Randbedingungen als auch die zu jedem Zeitpunkt möglichen Aktionen bekannt. Probleme, die traditionell mit dem Computer bearbeitet wurden, waren wohl definiert. Dieses Verständnis des Computers als Werkzeug zur Unterstützung repetitiver und mechanischer Arbeiten war lange Zeit ein taugliches Paradigma für seine Entwickler wie für seine Anwender. Neuere Anwendungen wie Computerspiele, künstlerische Installationen, VR-Welten, Lernsysteme, etc., lassen sich dagegen nicht zufriedenstellend als wohl definierte Probleme charakterisieren. Als Entwurfsprobleme gehören sie zum Gebiet der so genannten schlecht definierten Probleme. Im Zentrum jedes Entwurfsprozesses steht ein zweckgerichtetes Anliegen. Schwierig sind Entwurfsprozesse aufgrund ihrer Eigenschaft, dass das Anliegen sich während des Entwurfsprozesses durch seine Bearbeitung ändert. Freier Entwurf setzt immer einen Spielraum von Wahlmöglichkeiten und spielerisches Vorgehen voraus, um die unterschiedlichen Alternativen auszuloten. Entwurfsprozesse schlecht definierter Probleme zeichnen sich durch eine Reihe schwieriger Charakteristika aus:

- Moving Targets – Die Probleme sind nicht vollständig formalisierbar und die Ziele verschieben sich, während man am Problem arbeitet.
- Ein Großteil der Entwicklungszeit muss für die Problemfindung verwendet werden.
- Die Grenzen für die relevanten Informationen sind vage.
- Die Methoden zur Lösung des Problems sind nicht oder nur teilweise bekannt.
- Es gibt keine definitiven Kriterien, um erzielte Lösungen zu beurteilen.
- Die erzielten Lösungen tragen die Handschrift des Bearbeiters.
- Kommunikation ist primär – Lösungen können oft nur arbeitsteilig erreicht werden.

Das Problem der Modellbildung in der Softwareentwicklung lässt sich abschließend wie folgt zusammenfassen: „Floyd versteht Softwareentwicklung als Realitätskonstruktion. Dabei treffen die Beteiligten Unterscheidungen und Entscheidungen, um mit der Vielfalt der Möglichkeiten umzugehen. So entfalten sie einen Design-Raum, in dem die Anwendungswelt, die formale Methodenwelt und die technische Realisierungswelt verknüpft werden und technischer Entwurf sowie soziale Gestaltung aufgehoben sind. Für gemeinschaftlich gelingende Gestaltung ist das dialogische Prinzip maßgeblich, indem eine Vielfalt individueller Perspektiven aufgezeigt und gekreuzt werden.“[38] Im Gegensatz zur professionellen Softwaretechnik der Informatik wird Software im künstlerischen Kontext selten in großen Teams und für große Verkaufszahlen entwickelt, sondern ist vielmehr ein sehr individueller Prozess, der vollkommen unstandardisiert und in der subjektiven Auseinandersetzung mit dem technisch-künstlerischen Material stattfindet. Gleichzeitig sind die technisch realisierten Arbeiten immer in einen gesellschaftlichen und künstlerischen Kontext eingebunden und treten in einen Dialog mit aktuellen Verhältnissen und kulturellen Traditionen.

1.1.3 Programmieren im künstlerischen Kontext – eine erste Annäherung

„Ist Code Kunst? Kann Code Kunst werden? Oder ist Softwarekunst womöglich ein programmierter Selbstwiderspruch, weil die logischen Formeln und gezügelten Strukturen einer Maschinensprache das Gegenteil künstlerischer Feinheit und Freiheit sind? Wenn wirksame Kunst autonome Widerstandsnester in der zielstrebigen Konsumgesellschaft installiert, hat sie dann nicht besseres zu tun als ihr Publikum an die beschränkte Präsenz eines Desktop-Computers zu ketten? Die Antworten auf diese Fragen finden sich nicht in den Programmzeilen der Softwarekunstwerke, sondern in der traditionellen Kunst selbst. Deren Fotografien, Konzepte und Kataloge werden schon seit langem an Computern bearbeitet. Auch spiegeln sich die Manipulationstechniken des Computerdesigns und der Bildbearbeitungsprogramme in der Gestalt selbst konventioneller malerischer Oberflächen. Das eigentliche Material aber, mit dem sich Bilder filtern und verfremden lassen; die Apparaturen, mit denen Kunst geplant, inventarisiert und in Umlauf gebracht wird, bleiben der Kunst entzogen. Während kein Thema denkbar scheint, zu dem nicht schon ein Kunstwerk auf dem Markt gehandelt würde, blieb die Software, eine der wichtigsten Sprachformen der Gesellschaft, ein Territorium ohne Kunst. Softwarekunst ist deshalb zunächst gar keine Kunst. Sie ist Kritik an der Kunst.“

Gerrit Gohlke[39]

Technik im Allgemeinen und Programmierung im speziellen sind Projektionen menschlichen Denkens in die Umwelt. Die technisch materialisierten Gedanken begegnen uns als eigenständige Objekte wieder und wirken auf uns zurück. Dieser elementare Zyklus des Entäußerns und Rückwirkens, der bereits

38 Ebd., S. 24.

39 In: Künstlerhaus Bethanien (Hrsg.), Software Art – Eine Reportage über den Code, S. 3.

für einfache handwerkliche Werkzeuge gilt, nimmt für komplexe technische Systeme wie Programme vielschichtige Formen an. Software zeichnet sich gegenüber traditionellen Werkzeugen durch Eigenaktivitäten aus und kann selbständig ihr eingeschriebene Tätigkeiten vollziehen. Ein zufälliger Beobachter dieses Treibens würde zurecht vom Verhalten eines Programms gegenüber seiner Umwelt sprechen. Computergesteuerte Prozesse heben sich aber auch von den gewohnten Maschinenprozessen ab, weil ihr Verhalten nicht starr festgeschrieben, sondern frei konfiguriert ist und situationsabhängig variiert werden kann. In einer ersten Annäherung heißt Programmieren nichts anderes, als das Verhalten einer Maschine festzulegen, ohne dafür noch einmal in ihre innere Konstruktion eingreifen zu müssen. Programmieren ist gewissermaßen ein nicht-invasives, zeichenbasiertes Verfahren, um die Tätigkeit von Maschinen festzulegen. In höheren Programmiersprachen wie Java beschreiben wir Verhaltensmechanismen, indem wir Objekte mit bestimmten Eigenschaften und ihr Zusammenwirken mit anderen Objekten spezifizieren. Informell können wir Objekte als gewisse Einheiten auffassen, die entweder greifbar oder sichtbar, oder zumindest verstandesmäßig wahrnehmbar sind. Der Entwurf solcher Objekte in der Programmierung erfordert enorme Anforderungen an die Vorstellungskraft der Entwickler. Wichtig ist dabei, dass die menschliche Vorstellungskraft nicht an zurückliegende Erfahrungen oder konkrete Wahrnehmungen gebunden ist. Nach Heinrich Popitz[40] versetzt der menschliche „Überschuss an Vorstellung über alle Erfahrung uns in die Lage, unsere Erfahrungen im Horizont des Möglichen zu sehen als eine Möglichkeit unter anderen. Im Horizont des Möglichen verliert das, was passiert, den Schein des Notwendigen. Vorstellend können wir dem Wirklichen das Mögliche entgegensetzen. Unsere Vorstellungskraft ist ein Vermögen zur Alternative, ein Vermögen, Gegenwirklichkeiten zu entwerfen. Weil wir fähig sind, erfahrene Wirklichkeiten zu transzendieren, können wir kreativ neue Wirklichkeiten hervorbringen.“ Auf diese Weise ist der Mensch in der Lage, etwas Neues zu schaffen und die Bedingungen seiner eigenen Existenz zu verändern. „Er kann aus eigenen Kräften die Welt, in der er lebt, verändern und so als Urheber seine eigene Existenz neu definieren. Grundlegend und dauerhaft kann dies auf drei Wegen gelingen:

- durch erkundende Phantasie – suchend, probierend, fragend, entdeckend, erfindend – auf der Suche nach neuem Wissen,
- durch gestaltende Phantasie – Artefakte technisch und künstlerisch herstellend und formend – auf der Suche nach neuen Gehalten und Weisen des Bewirkens,
- durch sinnstiftende Phantasie – deutend, begründend, rechtfertigend – auf der Suche nach neuem Sinn.

40 In: Wege der Kreativität, S. 83.

Erkundende und sinnstiftende Phantasie sind kognitiv kreativ, insofern neues Wissen und neue Glaubensgewissheiten entstehen; gestaltende Phantasie wirkt faktisch, 'artefaktisch'."[41] Alle drei Formen der Kreativität sind auch im Umgang mit dem Computer notwendig, wenn seine Verwendung über die wirtschaftlich dominierten Fragen durchschnittlicher Softwareentwicklungen hinausreichen soll. Diese drei Kräfte der Phantasie müssen den Kontext komplexer und gelungener Programmentwicklung bilden. Im konkreten Prozess der Codierung selbst steht aber tatsächlich eine einfachere Leistung der Vorstellungskraft im Zentrum – die Antizipation von Handlungen. Programmieren heißt konkret, Handlungsanweisungen für einen Computer zu notieren. Hierbei müssen wir antizipierend die Folgen dieser Handlungen vorwegnehmen, nur so gelingt es, fehlerfreie Programme zu schreiben. Während bei mathematisch/technischen Anwendungen meist die formal/logische Antizipation gefragt ist, muss bei Anwendungen im ästhetischen und kulturellen Umfeld eine gesellschaftlich geprägte soziale Handlungskompetenz hinzukommen. Wir müssen nicht nur Vorstellungen über das Verhalten und das Zusammenspiel von Objekten entwickeln, sondern auch über die Wirkungen, die das Programm bei seiner Verwendung auf seine Benutzer und das gesamte Umfeld seines Anwendungsgebietes hat. Dies stellt weitere Anforderungen an den Softwareentwickler, die Fähigkeit zur Allozentrik. „Wir können der Welt mit allozentrischem Interesse begegnen. Wir können sie von dem her begreifen, was anders und außerhalb unserer selbst ist. Basis dieser Leistung ist die Wahrnehmungs-Ungebundenheit der Vorstellungskraft. Dass wir uns aus der gegebenen Wahrnehmungswelt lösen und vorstellend-phantasiebegabt in Verborgenes eindringen können, setzt uns zum Anderssein des Außer-uns-Seienden in ein besonderes Verhältnis. Wir erkunden, was uns verschlossen ist, wir gestalten, was jenseits der eigenen Existenz eine eigene Realität wird und produzieren so ein eigenständig-objektives Gegenüber, wir setzen uns mit dem verborgenen Nicht-Selbst numinoser Kräfte in Beziehung. Die vorstellungskräftige Phantasie transportiert uns über uns selbst hinaus in eine andere Realität, die wir als andere erkennen und anerkennen können."[42]

Heinrich Popitz bezieht sich hier auf die menschliche Kompetenz zum Perspektivwechsel nicht nur im Verhältnis zu Objekten, sondern vor allem in sozialen Beziehungen. Das Sichversetzen in den Interaktionspartner und die Fähigkeit, sich selbst vom Andern her zu sehen, erlernt bereits jedes Kind um die Wende des ersten Lebensjahres. Diese 'kognitive Schleife' ist Kennzeichen des Niveaus allozentrischer Kommunikation. Im dritten Abschnitt dieses Kapitels werden wir Programmierung als semiotischen Prozess beschreiben und dabei noch einmal verdeutlichen, dass Programmentwickler in der Lage sein müssen, permanent zwischen unterschiedlichen Perspektiven zu wechseln. Das Vorstellen findet in diesem Bild aber beim Menschen und nicht in der Maschine statt.

41 Heinrich Popitz, Wege der Kreativität, S. 3.

42 Ebd., S. 99.

Darauf legen wir in der hier vertretenen Auffassung der Programmierung großen Wert. Dies ist die Differenz zur künstlichen Intelligenz, die das Vorstellen in die Maschine verlagern will. Das wenige, das wir bisher zu den Anforderungen der Programmierung gesagt haben, sollte aber bereits deutlich machen, warum gute Programmierung schwierig ist, und warum die Märkte mit schlechter Software überschwemmt werden.

Die Computerwissenschaften haben sehr früh erkannt, dass Programmierung eine eigenständige neue Disziplin ist, die eigene Fragestellungen zu bewältigen hat. Zu den frühen grundlegenden Arbeiten zur Programmierung gehört Donald Knuths Buchreihe *The Art of Computer Programming*[43]. Das 1968 erstmals erschienene dreibändige Werk wird ohne Unterbrechung bis heute immer wieder neu aufgelegt, ein in der Informatik seltenes Ereignis. Knuth beginnt seine Arbeit mit der Feststellung: „The process of preparing programs for a digital computer is especially attractive, not only because it can be economically and scientifically rewarding, but also because it can be an aesthetic experience much like composing poetry or music." Bereits 1974 – sechs Jahre nach Erscheinen des ersten Bandes – bekommt Donald Knuth für seine Arbeiten den Turing Award der „Association for Computing Machinery" (ACM), die bekannteste Auszeichnung in den Computerwissenschaften. In seinem Vortrag anlässlich der Verleihung des Turing Awards erläutert er, warum er glaubt, dass 'Art' die richtige Bezeichnung für die Tätigkeit des Programmierens ist.[44] Als Hauptgrund führt Knuth an, dass Programmieren intellektuell und emotional befriedigend sein kann, weil es eine echte Errungenschaft in der Meisterung von Komplexität und der Erstellung von Systemen konsistenter Regeln darstellt. Heute widmet sich den Fragen des geeigneten Softwareentwurfs die Disziplin des *Software Engineering*. Mit Hilfe ihrer Methoden versucht man vor allem der so genannten Softwarekrise Herr zu werden. In den frühen 1960-er Jahren glaubte man noch, die Programmierung im Griff zu haben. Der Begriff 'Software' kam um ca. 1959-1960 in Gebrauch. Zu dieser Zeit war erst wenige Jahre allgemein akzeptiert, dass Software – komplementär zur Hardware – ein eigenständiges neues Feld mit eigenen Fragestellungen war. Innerhalb weniger Jahre, in denen Geschwindigkeit und Speicherkapazität der Maschinen um den Faktor 10 anstiegen, wurde das Debakel der Softwareentwicklung immer deutlicher. Man war software-technisch fähig, Programme mit 10.000 Zeilen Code zu schreiben, hatte aber Probleme mit Programmen, die 10-mal länger waren. Noch größere Projekte, mit einer Million Codezeilen, endeten im Desaster.[45] Die Schwierigkeiten bei der Herstellung von großen Software-Paketen wurden in Bezug auf die aufzuwendende Zeit, die Kosten und vor allem die Zuverlässigkeit immer offensichtlicher. Die bekannteste

43 Addison-Wesley Publishing Company, Reading 1968.

44 Donald E. Knuth, Computer Programming as an Art.

45 Vgl. Martin Campbell-Kelly and William Aspray, Computer – A History of the Information Machine, S. 196 – 200.

Katastrophe der jungen Softwaretechnik ereignete sich beim Raumschiff *Mariner I*, das am Morgen des 22 Juli 1962 startete, um zum Planeten Venus zu fliegen. Direkt nach dem Start geriet die Rakete außer Kontrolle und musste nach 290 Sekunden Flugzeit zerstört werden. Ein 18.5 Millionen Dollar Projekt scheiterte an einer einzigen falschen Programmzeile. Der 'Mariner I-Software-Bug' wurde zur Ikone der Softwarekrise. Bis heute ist der Versuch, die Komplexität großer Softwaresysteme in den Griff zu bekommen, eines der Kernthemen der praktischen Informatik.

Programmierung im künstlerischen Kontext unterscheidet sich schon in der Herangehensweise und den Erwartungen wesentlich von der Softwaretechnik der Informatik. Zwar sind auch die Sichtweisen der Softwaretechnik sehr breit angelegt und beschränken sich durchaus nicht auf Software als rein technisches Artefakt, sondern sehen Software als sozial eingebettete Systeme, die zunächst in einem gleichfalls sozialen und kommunikativen Prozess hergestellt werden müssen. Software wird insbesondere auch als Arbeitswerkzeug und Kommunikationsmedium betrachtet, das ergonomisch, d.h. menschengerecht gestaltet werden muss. Aber Softwaretechnik ist, im Gegensatz zur künstlerischen Programmierung, ein stark standardisierter Prozess und viel stärker an kommerziellen Kriterien orientiert. Softwaretechnik (engl. *Software Engineering*) befasst sich mit der ingenieursmäßigen Konstruktion und der professionellen Entwicklung großer Softwaresysteme, d.h. es stehen komplexe, qualitativ hochwertige Produkte und deren zuverlässiges Funktionieren im Vordergrund. Als Qualitätsmerkmale gelten neben Preis und Bedienerfreundlichkeit die Korrektheit, Effizienz, Zuverlässigkeit und Anpassbarkeit des Systems. Das Produkt Software muss in erster Linie wettbewerbsfähig sein. Software wird dabei als formalsprachliches Gebilde aufgefasst, das aus der zu Beginn erarbeiteten Spezifikation des Produktes abgeleitet wird. Da große Softwarepakete im Zentrum der Softwaretechnik stehen, die von Teams mit bestimmten zeitlichen Vorgaben entwickelt werden müssen, sind Aspekte der Arbeitsteilung, der Zusammenarbeit und der Komplexitätsreduktion von besonderem Interesse. Der Herstellungsprozess wird hierzu anhand von Vorgehensmodellen in einzelne Phasen mit strikten Qualitätskontrollen gegliedert. Organisation und Management von Softwareentwicklungsprojekten sind ein wesentliches Erfolgskriterium für kommerzielle Softwareprodukte. Ganz anders die Situation der Programmierung als künstlerische Praxis. Der Computer wird hier als Werkzeug der Imagination eingesetzt. In der Kunst geht es im Unterschied zur Technik nicht vorrangig um die Lösung a priori definierbarer Probleme, sondern um die Annäherung an Problemstellungen und die sukzessive Fixierung von Fragen. Es existiert im Unterschied zu mathematisch/technischen Problemen auch keine Lösung, die nach allgemein akzeptierten Kriterien zu bewerten wäre. Es gibt keinen festzulegenden Punkt, an dem ein Projekt eindeutig und endgültig abgeschlossen wäre. Vielmehr wird die Entwicklung in einem zufrieden stellenden Status beendet, oder schlicht abgebrochen, weil ein Ausstellungstermin

unaufhaltsam naht. Die direkte Auseinandersetzung mit dem Code als Material und die Widerstände, die sich erst im Vorgang des Programmierens zeigen und überwunden werden müssen, sind hierbei genauso wichtig wie das Endprodukt. Der Prozess der Softwareerstellung ist in künstlerischen Zusammenhängen sehr viel persönlicher, die Sicht- und Herangehensweisen sind subjektiver. Es ist nicht wie in der Informatik möglich, zunächst eine Spezifikation der Software vorzulegen, die es dann erlaubt, die eigentliche Programmerstellung den Codierern zu überlassen. Programmierende Künstler begreifen den Code als Material und damit als wesentliche Komponente ihrer Projekte. Sie können sich nicht vorstellen, diese Aufgabe ohne Authentizitätsverlust für ihre Arbeiten an andere zu delegieren. Von Künstlern entwickelte Software ist nie ganz fertig, sie bleibt stets bis zu einem gewissen Grad unscharf und offen und enthält immer die Aufforderung, weiter über sie nachzudenken und sie zu verändern. Sobald dieser Prozess der permanenten Metamorphose – sowohl der Ideen als auch des Codes – zum Erliegen kommt, wird das Programm überflüssig. Am Ende der Entwicklung steht deshalb in der Regel nicht der Verkauf der Software, sondern entweder der Mülleimer oder das partielle Überleben einzelner Module im Rahmen neuer Projekte.

Medientheorie und Medienkunst haben seit ca. 20 Jahren Konjunktur. Beide scheinen sich sowohl in den Institutionen wie im gesellschaftlichen Bewusstsein etabliert zu haben und gelten als Leitfiguren von Wissenschaft und Kunst. Medienkunst präsentiert sich dabei in aller Regel als Behauptung des Neuen und Aufhebung der Tradition. Sie versteht sich als technische Ermöglichung zentraler Utopien der Avantgarde.[46] Obwohl es im praktischen Umgang mit Programmcodes und den Zielsetzungen der Programmierung einen Unterschied zwischen Wissenschaftlern und Künstlern gibt, existieren tieferschichtige Gemeinsamkeiten. „Die letzte und tiefste Gemeinsamkeit von künstlerischer Invention und technischer Erfindung liegt in ihrem Entwurfscharakter. Beide stellen etwas vor, das denkbar, aber noch nicht machbar ist. Der Nachweis der Machbarkeit unterliegt dann allerdings extrem verschiedenen Bewertungsmaßstäben: in der Kunst dem des ästhetischen Gelingens, in der Technik dem der realen Funktion. Deshalb mündet Technikentwicklung in eine Eigendynamik, die wie ein Maschinendarwinismus alles nicht Wettbewerbsfähige ausscheidet. Ihr Motor ist das positivistische Ideal permanenter Innovation und ihr Treibstoff der kommerzielle Erfolg. Deshalb bleibt für Ziele und Motive der Erfinder kein Platz mehr. ... Als Gegenmodell zur Apologie der universellen Machbarkeit durch die Technik bildet die uneingelöste und unerfüllbare Utopie laut Adorno ein philosophisches Kriterium zur Definition der Rolle der Künste.“[47] In der Folge macht Daniels aber auch deutlich, dass künstlerische Entwürfe zur Nutzung und Weiterentwicklung von

46 Umfassende Betrachtungen und kritische Analysen zum Verhältnis von Kunst, Wissenschaft, Technologie und Lebenswelt wie auch zum Begriff der ‘Medienkunst’ finden sich in den Arbeiten von Hans Ulrich Reck. Siehe Literaturverzeichnis.

47 Dieter Daniels, Kunst als Sendung, S. 14.

Medien ganz anders in Erfüllung gehen können. Sie verwirklichen sich nämlich nicht zwangsweise im Kontext der Künste, sondern in dem der Massenmedien und verlieren dabei durchweg ihre utopische Dimension.

Programmierung ist Begegnung mit sich selbst und mit entäußerten Gedanken und Vorstellungen. Ähnlich dem literarischen Schreiben werden Gedanken in Reihenfolge gebracht und schriftlich festgehalten. Den niedergeschriebenen, entäußerten Gedanken begegnen wir im Falle eines normalen Textes beim Lesen wieder. In der Programmierung begegnen wir unseren Vorstellungen in der Ausführung in Form von Prozessen oder errechneten Ergebnissen wieder. Wenn man aber in diesem Prozess die Prinzipien des eigenen Denkens ungedacht lässt, ist man nur Werkzeug dessen, was man zu denken meint. Die Geschichte der Programmierung ist deshalb eine Geistesgeschichte des Nachdenkens über das Denken. So wie allerdings das Denken einen Gegenstand braucht, um sichtbar und analysierbar zu werden, brauchen Programmiersprachen eine Anwendung, um ihre eigene Struktur und ihren Nutzen sichtbar werden zu lassen. Kein Denken ohne Gegenstand – keine Programmierung ohne Anwendung. Der Computer ist eine neue technische Hybridform aus strikten funktionalen Grundelementen und einer unausgeloteten Offenheit, die durch die Programmierbarkeit der Maschine entsteht. Im Programmcode fallen Alphabetisierung und Mechanisierung in eins, er ist sprachliche Beschreibung und Arbeitsanweisung zugleich. In der Ausführung eines Programmtextes schlägt die Schriftform des Programms um in Handlungen und Prozesse. Ähnlich der Schriftform natürlicher Sprachen sind die syntaktischen Elemente von Programmiersprachen zwar starr und vorgegeben, dennoch ist das, was sich mit ihnen ausdrücken lässt unbegrenzt und universell. Erst diese Universalität und Offenheit – die im Verlauf des Buches noch stärker herauszuarbeiten ist – macht den Computer zu einen interessanten Arbeitsmittel für die Kunst. Der Programmtext muss aber im Unterschied zum literarischen Text technisch funktionieren, er muss lauffähig sein. Deshalb unterliegt in der Praxis die generelle Machbarkeit eines Entwurfs und die tatsächliche Form seiner Realisierung, gleichviel ob es sich um einen künstlerischen oder technischen Programmentwurf handelt, stark den Organisationsstrukturen der Maschine und den darauf aufsitzenden Programmiersprachen. Fragen des technischen Herstellens und Funktionierens müssen in der künstlerischen Programmierung mit den Fragen des ästhetischen Gelingens in Übereinstimmung gebracht werden, sie sind nur zusammen zu lösen. Die innere Organisationsstruktur des Computers wirkt so am künstlerischen Produkt mit.

Eines der dominierenden Leitbilder des Computereinsatzes in künstlerischen Zusammenhängen ist die Partizipation des Betrachters. Zwar stellt sich diese Frage schon lange vor der Verwendung avancierter Technologien in der Kunst, aber erst durch programmierbare Medien kann sich auch die logische und ästhetische Struktur des Kunstwerks verändern, und zwar überprüfbar und nicht nur in der Interpretation des Betrachters. Das Kunstwerk soll zu

einem kollaborativen Vorgang werden, an dem sich Künstler, Rezipienten und nicht zuletzt die Maschine beteiligen. Methoden, die eine Eigendynamik von Programmen unterstützen, d.h. die offen sind und sich in der Interaktion mit dem Benutzer verändert, sind von besonderem Interesse. Diese Offenheit von Programmen, sich jenseits vorstrukturierter Entscheidungspfade zu entwickeln, ist eine der größten Herausforderungen für interaktive Medien. Ein anderer Bereich, der in der theoretischen Reflexion im Vergleich zur Interaktivität neuer Medien bisher wenig untersucht worden ist, sind die 'Interfaces'. Unter dem Begriff Interface verstehen wir hier den Ort der Begegnung zwischen der Umgebung, in die ein System eingebettet ist, und seiner inneren Struktur. Im Interface findet eingabeseitig die Reduktion der Umwelt auf alphanumerische Zeichen und ausgabeseitig die Rücktransformation von Zeichen in unscharfe, für den Benutzer wahrnehmbare Reize statt. Das Interface verbindet Wahrnehmung und Kognition, die bei gelungenen Arbeiten einem gemeinsamen konzeptuellen Raum entspringen und eine untrennbare Einheit bilden. Der Abbildungsprozess in den Interfaces ist für die künstlerische Praxis mit neuen Medien von mindestens gleicher Bedeutung wie der Verarbeitungsprozess der Zeichen selbst. Die Einschränkung des Schnittstellenbegriffes auf ergonomische Fragestellungen reicht im künstlerischen Kontext nicht aus. Insgesamt jedoch beschränkt sich der Umgang mit Programmcodes in der Kunst keineswegs auf die Partizipation des Betrachters.

Wer die Sprache des Codes zu lesen in der Lage ist, dem steht das Medium Computer in seiner ganzen Multiperspektivität offen, die weit über den bloßen Werkzeuggebrauch hinausreicht und sich auch nicht in den ewigen Variationen bekannter Wahrnehmungsformen erschöpft. Erst innerhalb des Horizontes der Codes können alle künstlerischen Strategien und Herangehensweisen konsequent durchgespielt werden. Entgegen der allgemeinen Vorstellung lehnen wir die Sicht auf den Code als abstrakten, rationalen, objektivierten und stringenten Gegenstand ab. Vielmehr treten in der Programmierung das Abstrakte und das Konkrete, das Universelle und das Besondere, das Subjektive und das Objektive in einen Dialog. Wir können gezielt Störungen einbauen oder das Medium dekonstruieren, um die Eigenstrukturen sichtbar zu machen oder den konstruktiven Charakter des Mediums in den Vordergrund zu stellen. Wir können den Code selbst zum Material erklären und diesen in seiner Prozesshaftigkeit oder seiner Zeichenhaftigkeit, oder beides zusammen untersuchen. Mit dem Code steht uns auch das ganze Repertoire der Mittel zur Verfügung, um den Mediengebrauch in sich zu verändern und zu erweitern. Wir können mit der Maschinenhaftigkeit des Mediums spielen und diese hervorheben oder umgekehrt die Mechanik dissimulieren. Wir können neue Ästhetiken untersuchen, die sich wahlweise auf die Produkte beziehen, also auf die erzeugten Bilder, Klänge und Prozesse, oder – im Sinne der Konzeptkunst – auf die dahinter stehenden abstrakten Ideen und Methoden. Ästhetische Fragen, die sich auf den Code selbst beziehen, können genauso untersucht werden wie Fragen des

Stils der Programmierung. Überlegungen zur Objektivierung und der Entledigung des Individuellen in der Kunst – wie sie die Konzeptkunst vertritt – lassen sich ebenso anstellen wie Untersuchungen zur Rationalität und Irrationalität bzw. zur Schärfe und Unschärfe formaler Prozesse. Der Code muss dazu weder auf einer Maschine ausgeführt werden, noch muss er überhaupt lauffähig sein. Erst wenn Künstler die mächtige Form der Programmierung beherrschen, können sie beginnen, ihre eigenen Werkzeuge zu entwickeln. Nur wer selbst Programmierer ist, kann auch die Folgen wirtschaftlicher und politischer Vorstöße, etwa im Bereich des 'Digital Rights Management' (DRM) oder des 'Trusted Computings' in letzter Konsequenz einschätzen. Soweit nur einige Argumente, die alle nicht mehr wollen, als unseren Wunsch nach einer radikalen Perspektivität in Bezug auf Programmcodes zu unterstützen. Die Vielfalt möglicher Sichtweisen greift weit über die übliche Frage der Softwaretechnik hinaus, die sich noch immer wesentlich darauf konzentriert, wie man ein einzelnes Programm vernünftig zum Laufen bringt. Softwarekunst, algorithmische Kunst, Konzeptkunst, Generative Kunst, aber auch Game Art und Netzkunst sind Felder, die sich mit Fragen des Codes in der Kunst beschäftigen. Das Anliegen des Buches ist nicht, direkt zu diesen Fragen Stellung zu nehmen, sondern vielmehr die Grundprinzipien der Programmierung deutlich zu machen und damit Software überhaupt erst als Reflexionsgegenstand verfügbar zu machen. Wir begreifen Software als kulturell und gesellschaftlich codiertes Mittel, das heute eingebettet ist in ein dominantes kommerzielles System. Aber erst durch die Fähigkeit, den Code zu lesen und durch das Wissen um die dahinter stehenden Denktraditionen werden die Grundbedingungen geschaffen, um in künstlerische, gesellschaftliche und politische Diskurse eingreifen zu können.

```
Perl Routines to Manipulate London w.blake@scotoma.org
# London.pl, v 0.0.0 1792/02/23 08:16:43
# UNFINISHED
# Copyright (c) 1792-2002 William Blake
# Unpublished work reconstituted W.Blake@Scotoma.org.

SYNOPIS and DESCRIPTION
# This Library is for redressing the gross loss to Londons Imagination of children
# beaten enslaved fucked and exploited to death from 1792  to the present.
# We see this loss in every face marked with weakness or marked with woe.

use PublicAddressSystem qw(Hampstead Westminster Lambeth Chertsey);
# PublicAddressSystem is an I/O library for the manipulation of the wheelen
# Vortex4 129db outside warning system.
#
# from Hampstead in the North, to Peckham in the South,
# from Bow in the East to Chertsey in the West.

# Find and calculate the gross lung-capacity of the children screaming from 1792 to the present
# calculate the air displacement needed to represent the public scream
# set PublicAddressSystem intance and transmit the output.
# to do this we approximate that there are 7452520 or so faces that live in the charter'd streets of
London.
# Found near where the charter'd Thames does flow.
#
# DATATYPES USED:

 local @SocialClass = qw(
        RentBoy
YoungGirl-Syphalitic-Innoculator
        CrackKid WarBeatenKid
ForcedFeatalAbortion
        Chimney-Sweeps
UnCategorisedVictim

        );

#
#       These are a series of anonymous hashes;
#       At least one is required at compile time:
#

local %DeadChildIndex;

#       The Data for the DeadChildIndex should be structured as follows:
#
#       %{DeadChildIndex} => {
#               IndexValue => {
#                       Name            =>      " Child name If known else undefined ";
#                       Age             =>      " Must be under 14 or the code will throw an
#                               exception due to $COMPLICITY";
#                       Height          =>      "Height of the child"
#                       SocialClass     =>      "RentBoy YoungGirl-Syphalitic-Innoculator
#                                               CrackKid WarBeatenKid ForcedFeatalAbortion
#                                               Chimney-Sweeps UncategorisedVictim "
#               }, As many as found
#       }
#

#       CryOfEveryMan
#       First we add the Class attribute to the DeadChild instance under review
#       Next add the VitalLungCapacity of that childs ability to scream

        sub CryOfEveryMan {
                my $index = shift;
                # Because a child may belong to one or more SocialClass
                # traverse the list adding the prospects of that SocialClass
                foreach my $Class (@SocialClasses){
                    # Add the contents of this $Class to $DeadChildIndex->{$Index}
                    # Class attribute
                    if( $Class eq $DeadChildIndex->{$Index}->{Class}){
                        $DeadChildIndex->{$Index}->{Class} = %{$Class} ;
                    }else{
                        warn "$DeadChildIndex->{$Index}->{Name} is not a member of = $Class\n";
                    }
                }
```

Abbildung 4:
Graham Harwood, *London.pl.*[48] Eine Zusammenstellung von Auszügen aus dem Gesamttext. Das Codefragment ist die Verarbeitung eines Gedichtes von William Blake über die sozialen Verhältnisse von 1796. Die Syntax der Programmiersprache PERL wird nicht benutzt, um Computerbefehle zu formulieren, sondern zu Elementen der Alltagsprache umfunktioniert. Das PERL-Skript ist als Programmcode nicht lauffähig.

48 http://www.scotoma.org/lungs/london.txt.

1.2 Die Struktur der programmierbaren Maschine

„Zufälligerweise ist Programmieren eine Kunst, die relativ leicht zu erlernen ist. Fast jeder, der zu einem sauberen methodischen Denken imstande ist, kann durch geringe Unterweisung und mit einiger Übung ein recht guter Programmierer werden. Und da das Programmieren fast unmittelbar zu Belohnungen verhilft, d.h. weil ein Computer sich sehr schnell in etwa der Weise verhält, wie der Programmierer es beabsichtigt hat, ist das Programmieren besonders für Anfänger sehr verführerisch. ... Wenn Studenten mit dem Studium [einer bloßen Verfeinerung der Programmiertechnik] fertig sind, gleichen sie Leuten, die es auf irgendeine Weise fertiggebracht haben, sich flüssig in einer Fremdsprache auszudrücken, nur um dann festzustellen, dass sie gar nichts Eigenes zu sagen haben."

Josef Weizenbaum[1]

Im Allgemeinen wird der Computer als ein Produkt der Neuzeit charakterisiert, die mit Galilei, Descartes und Leibniz einsetzt. Wir wissen heute, dass dies im engeren Sinn und im Hinblick auf die formale Struktur der Universalmaschine zwar richtig ist, das Konzept der externen Steuerung des Verhaltens von Maschinen allerdings bereits lange davor in der Mechanik erfunden war. Auch algorithmische Verfahren waren seit der Antike bekannt – wenn auch nicht unter dem Begriff Algorithmus – und hatten im so genannten Heron-Verfahren (sumerisch-babylonisches Wurzelziehen), im Euklidschen Algorithmus (Bestimmung des größten gemeinsamen Teilers zweier Zahlen), dem Sieb des Eratosthenes (Bestimmung von Primzahlen) und dem Verfahren von Archimedes (Approximation der Kreiszahl π) bereits vier bis heute gültige Klassiker hervorgebracht. Auch die Algebraisierung, das Rechnen mit Buchstaben und Symbolen, hatte bereits einen bemerkenswerten Stand erreicht. Die mathematisch-naturwissenschaftliche Naturauffassung führte zur konsequenten Weiterentwicklung der Formalisierung und ihrer Anwendung auf die gesamte Lebenswelt, die natürlichen Erscheinungen wie auch die technischen Artefakte. Diese umfassenden Anstrengungen kulminieren in der programmierbaren Maschine – allerdings zu einem Zeitpunkt, zu dem die philosophischen und wissenschaftlichen Standpunkte, die diese Maschine hervorgebracht haben, längst überholt sind.

Der folgende Abschnitt gibt im ersten Teil einen kurzen, exemplarischen Einblick in die Entwicklungsgeschichte der programmierbaren Maschine. Es wird dabei nicht um eine umfassende und lückenlose Darstellung der Geistesgeschichte der Programmierung gehen, sondern vielmehr darum – auch im Zusammenwirken mit den Zwischenspielen der Kapitel 2 und 3, die manches von dem hier Angedeuteten vertiefen – durch Herausgreifen einzelner einschneidender Entwicklungen und bedeutender Persönlichkeiten, eine Sensibilität für die geistige

1 In: Die Macht der Computer und die Ohnmacht der Vernunft, S. 362 f.

Tradition der programmierbaren Maschine zu erreichen. Wir werden sehen, dass heutige Maschinen – Computer im Speziellen – formale Konstruktionen sind, die von ihrer materiellen Substanz ebenso abstrahieren wie von den verwendeten Antriebsenergien. Vor allem soll im Verlaufe der Ausführungen deutlich werden, dass Maschinen nicht unabhängig vom Menschen betrachtet werden können. Wir wollen in Anlehnung an Bammé et. al.[2] Maschinen als materialisierte Projektionen von Wesensmerkmalen des Menschen begreifen. Diese Auffassung hat weitreichende Konsequenzen: „Wenn Sozialwissenschaftler (vor allem im deutschen Sprachraum) einwenden, hierbei handle es sich um bloße Analogien, um Metaphern, um mehr nicht, so scheinen sie auf den ersten Blick recht zu haben. Und doch irren sie. Denn ihnen schwebt, Folge deutscher Kulturtradition mit Berührungsängsten vor naturwissenschaftlichen und technologischen Entwicklungen, immer noch der klassisch-mechanische Maschinenbegriff vor, wie sie ihn im Physikunterricht der Schule gelernt haben. Entgangen ist ihnen, dass die Bedeutung des Maschinenbegriffs sich in der Zwischenzeit deutlich verändert hat – wie auch die Erscheinungsform der Maschine selbst. Der klassisch-mechanische Begriff orientierte sich eng an der Zweckbestimmung der einzelnen Maschine. Heute rückt die Kategorie des 'Verhaltens' ins Zentrum der Diskussion. Diese Akzentverschiebung gegenüber dem klassischen Begriff wurde notwendig, um die materiellen und immateriellen Bestandteile, zum Beispiel eines Computers, als Maschine, was er zweifellos ist, erfassen zu können."[3] Die Autoren gehen in ihrem Buch *Maschinen-Menschen Mensch-Maschinen* des weiteren davon aus, dass es ein gemeinsames Drittes gibt, das sowohl den Menschen als auch die Maschine prägt. „Dieses gemeinsame Dritte sind historisch-gesellschaftliche Strukturprinzipien, die, vermittelt über ständig sich wiederholende Interaktionen, zur Psychostruktur sedimentieren und sich in Maschinerie und Organisationsstruktur vergegenständlichen. Psychostruktur, Organisationsstruktur und Maschinenstruktur sind versteinerte Formen sozialer Beziehungen."[4] Anders herum könnte man sagen, es ist die Gesellschaft, die in Form des abgespaltenen Intellekts denkt. Da Programmieren nichts anderes heißt, als bestimmte Arten von Maschinen zu konstruieren, die frei von allem sind, was außerhalb des Denkens und der Vernunft liegt, finden sich diese historisch-gesellschaftlichen Grundstrukturen – als Sedimente – natürlich auch in unseren Programmiersprachen und Programmen wieder.

2 Arno Bammé, et. al., Maschinen-Menschen Mensch-Maschinen.

3 Ebd., S. 112.

4 Ebd., S. 110.

1.2.1 Unsichtbare Maschinen

„Die Schwierigkeit bestand darin, eine zufällige Ansammlung von Menschen, losgelöst von Familie und Gemeinschaft sowie von ihren vertrauten Beschäftigungen, jeder mit eigenem Willen oder zumindest der Erinnerung an einen solchen, in eine mechanisierte Gruppe zu verwandeln, die auf Befehl manipuliert werden konnte. Das Geheimnis der mechanischen Kontrolle bestand darin, dass ein einziger Kopf mit genau bestimmtem Ziel an der Spitze der Organisation war und dass es eine Methode gab, Anweisungen über eine Reihe von Funktionären weiterzugeben, bis sie die kleinste Einheit erreichten. Exakte Weitergabe der Anweisung und absolute Unterwerfung waren gleichermaßen von wesentlicher Bedeutung.“
Lewis Mumford[5]

In seiner umfangreichen Arbeit *Mythos der Maschine – Kultur, Technik und Macht*[6] stellt Lewis Mumford die These auf, dass die Bedeutung der Werkzeuge für die Entwicklung des Menschen massiv überschätzt wird. Für das Überleben des Frühmenschen war es seiner Ansicht nach weniger wichtig Werkzeuge zu gebrauchen, als vielmehr sich Wissen über die Natur anzueignen. Für Mumford ist die Entwicklung von Kultur und Technik lediglich Ausdruck eines geistigen Überschusses, der charakteristisch für den Frühmenschen ist und ihn damit vom Tier unterscheidet. In seinen Augen sind für die Entwicklungen der Technik weniger technische Erfindungen ausschlaggebend gewesen, als vielmehr die Entstehung sozialer und organisatorischer Systeme. Erwähnenswert ist in diesem Zusammenhang, dass Mumford die Rede von der industriellen Revolution ablehnt. Für ihn ist die technische Entwicklung ein kontinuierlicher Prozess der mit dem frühen Menschen in den neolithischen Dörfern beginnt und sich stetig bis in unsere Zeit fortsetzt. Zum Zeitpunkt der so genannten industriellen Revolution waren die meisten Techniken schon erfunden (Wind- und Wasserkraft) und neue Entwicklungen wie die Dampfmaschine, haben sich erst danach durchgesetzt. Mumford unterscheidet zwischen der demokratischen Polytechnik, die vor allem in mittelalterlichen Städten zu finden ist, und der autoritären Monotechnik, die das Kennzeichen der Megamaschine ist. Als Beispiele führt Mumford den ritualisierten Tagesablauf in benediktinischen Klöstern und die Organisation der städtischen Hochkulturen Mesopotamiens und Ägyptens im dritten Jahrtausend vor Christus an.

5 In: Mythos der Maschine, S. 223.

6 In der Originalausgabe erschienen unter dem Titel *The Myth of the Machine*, Volume I: Technics and Human Development (1967), Volume II: The Pentagon of Power (1970).

Abbildung 5:
Assyrer bilden einen Hügel, um einen Palast zu errichten. Bemerkenswert sind die Friese mit Soldaten. Sowohl Soldaten als auch Aufseher waren wichtige Elemente der Megamaschine.

Die Erfindung der Maschine geht nach Mumford auf die Hochkultur des Gottkönigtums in den frühen Perioden des Pyramidenzeitalters zurück. Es handelt sich dabei um einen Vorläufer jener mechanischen Maschinen, die dann erst Jahrtausende später in Stahlform das Bild der Industrialisierung geprägt haben. Diese archetypische Maschine, die sich in den frühen städtischen Hochkulturen herausbildet, besteht in ihren elementaren Bestandteilen aus räumlich vollkommen getrennten Einheiten, nämlich Menschen. Deshalb war sie lange unsichtbar und nicht als Vorläufer unserer heutigen Maschinen erkennbar und blieb daher auch namenlos. Trotzdem hat sie als Arbeitsmaschine über eine enorme Produktionskraft verfügt. Ihre Arbeitsleistung dokumentiert sie bis heute mit ihren Produkten, unter anderem der großen Pyramide von Gizeh. Mit Mumford:[7] „Diese

7 Ebd., S. 219.

außergewöhnliche Erfindung erwies sich tatsächlich als das früheste Arbeitsmodell für alle späteren komplexen Maschinen, obwohl sich der Schwerpunkt mit der Zeit von den menschlichen Bestandteilen auf die verlässlicheren mechanischen verschob. Die einzigartige Leistung des Königtums bestand darin, das Menschenpotential zu konzentrieren und die Organisation zu disziplinieren, was die Ausführung von Arbeiten in nie zuvor dagewesenem Ausmaß ermöglichte. Als Ergebnis dieser Erfindung wurden vor fünftausend Jahren technische Aufgaben bewältigt, die sich mit den heutigen Höchstleistungen der Massenproduktion, der Standardisierung und der präzisen Planung messen könne." Zu den Voraussetzungen der Megamaschine gehören verschiedene Submaschinen, die als integriertes Ganzes zusammenwirken und nur als Einheit ihre Leistungsfähigkeit entfalten. Elementare Submaschinen sind: die Religion des Sonnengottkönigtums, der Priesterkaste mit ihrem astronomischen Geheimwissen, die Bürokratie als Befehlszergliederungs- und Befehlsverteilungsmechanismus, die Militärmaschine als Unterdrückungs- und Machterweiterungsapparat und schließlich die Arbeitsmaschine, als ihre hochgradig arbeitsteilige Produktionseinheit. Die Megamaschine kann für unterschiedliche Zwecke eingesetzt werden, so produziert sie als Arbeitsmaschine riesige Überschüsse, die in riesigen Speichern gelagert und von der Bürokratiemaschine verteilt werden. Als Arbeitsmaschine errichtet sie auch mit Unterstützung der anderen Maschinenkomponenten in Form der Pyramiden gigantische Artefakte zur Verherrlichung der Religion und des Gottkönigs. Als Militärmaschine wird sie für Werke der Unterdrückung, Eroberung und Zerstörung eingesetzt.

Lewis Mumford ist auch im Zusammenhang mit unserem Thema, der Verbindung von Programmierung und künstlerischer Praktik interessant, weil er die heutige Trennung von Kunst und Technik für eine moderne Fehlleistung hält.[8] „Bevor die Maschine unsere ganze Aufmerksamkeit in Anspruch nahm, bestand eine kontinuierliche Wechselwirkung zwischen quantitativer Ordnung und Arbeitseffizienz einerseits und qualitativen Werten und Zwecken, die die menschliche Persönlichkeit widerspiegeln, andererseits. ... Schon die oberflächlichste historische Betrachtung der Künste enthüllt eine Fruchtbarkeit in der Erfindung von Formen, die bis zum neunzehnten Jahrhundert in der militärischen Maschinentechnik nicht ihresgleichen hatte; und die Entwicklung neuer ästhetischer Formen stellte neue Ansprüche an den technischen Erfindungsgeist, wie in der langen Reihe der Textilerfindungen, historisch beginnend bei den Webern von Damaskus über die Teppichweberei des Mittelalters bis zu den komplizierten ornamentalen Mustern, die den Anstoß zum Jacquard-Webstuhl gaben; dieser unterstreicht übrigens meine frühere Behauptung, denn die komplexe Lochkartensteuerung des Jacquard-Stuhls diente als Modell für die spätere Erfindung von Sortier- und Rechenmaschinen. ... So waren nicht nur

8 Ebd., S. 290 f.

ästhetische, sondern auch mechanische Erfindungen zum Zweck der Erzielung oder Vervollkommnung rein ästhetischer oder symbolischer Ergebnisse charakteristisch für einen großen Teil der noch nicht automatisierten Produktion. Dieser Beitrag wurde selbst in seinen technischen Implikationen von jenen unterschätzt, die Technik auf eine Eroberung von Zeit, Raum und Energie zu reduzieren pflegen."

Abbildung 6:
Pyramidenbau, von Gustav Richter, zeigt Ägypten im 3. Jahrtausend v. Chr. Im romantisierenden Stil des 19. Jahrhunderts gibt das Gemälde eine eindrucksvolle Sicht auf die 'Baustellen' im alten Ägypten. Herodot: „Und es arbeiteten je zehnmal zehntausend Mann ..." Über Hunderte von Kilometern mussten aus entfernten Steinbrüchen Tausende von tonnenschweren Steinblöcken herantransportiert werden.

Wenn wir Mumfords Megamaschine tatsächlich als Maschine anerkennen, dann ist die sich direkt anschliessende Frage nach der Programmierbarkeit dieser Maschine ebenfalls gerechtfertigt. In der Tat müssen wir zugestehen, dass der Zweck, dem die Megamaschine dient, nicht schon durch ihre Konstituierung festgelegt ist, sondern sie kann verhältnismäßig leicht für neue Aufgaben 'umprogrammiert' werden. In letzter Instanz gelenkt wird die Maschine durch den Herrscher, der Befehle an die Maschine gibt. Auf seinen Befehl hin führt die Maschine unterschiedliche Aufgaben aus. Sie kann im Überfluss Agrarprodukte herstellen, Pyramiden bauen oder als Militärmaschine Kriege führen. Gegebenenfalls wird die Maschine dafür von innen heraus ihre eigene Struktur adaptieren, um die Aufgabe überhaupt, oder besser erfüllen zu können. „Nur Könige, unterstützt von der Disziplin der astronomischen Wissenschaft und von den Geboten der Religion, waren imstande, die Megamaschine zusammenzufügen und zu lenken. Dies war ein unsichtbares Gebilde, zusammengesetzt aus lebenden, aber stabilen menschlichen Teilen, jeder für eine spezielle Funktion, Rolle und Aufgabe bestimmt, um die immense Arbeitsleistung und die grandiosen Pläne dieser riesigen kollektiven Organisation zu ermöglichen."[9] Dabei macht sich die Megamaschine ein Prinzip zu nutze, das auch in allen komplexen mechanischen

oder elektronischen Maschinen zu finden ist, das Prinzip der sukzessiven hierarchischen Unterteilung.[10] Eine Gesamtaufgabe wird nach dem Prinzip 'Teile und Herrsche' in kleinere, weitestgehend unabhängige Teilaufgaben zerlegt. Die so entstehenden Subsysteme werden wieder nach dem gleichen Prinzip darauf untersucht, wo Teilaufgaben sinnvoll heraus gelöst werden können. Wichtiges Kriterium für die Zerlegung ist zum Beispiel, dass die Interkommunikation zwischen den Subsystemen einer Ebene geringer ist als die Intrakommunikation, d.h. die Kommunikation innerhalb eines Subsystems. Auf den unteren Ebenen der mehrstufigen Zerlegung steht der Vorarbeiter, der nur noch über wenige Arbeiter herrscht und nicht mehr die Gesamtzusammenhänge sieht, d.h. das Zusammenwirken der Subsysteme auf den darüber liegenden Schichten. Diese Maschine zeigt nicht nur unter dem Gesichtspunkt der Programmierbarkeit eine erstaunliche Flexibilität. Sie besitzt Eigenschaften, die von Informatikern für die Computertechnologie seit Jahrzehnten angestrebt, bisher aber nur sehr eingeschränkt erreicht wurden. So arbeitet die Megamaschine zum Beispiel massiv parallel und zeigt dabei noch eine enorme 'Fehlertoleranz', d.h. beim 'Ausfall einzelner Elemente' – es sind beim Pyramidenbau Tausende von Menschen auf grausame Weise ums Leben gekommen – erweist sich die Organisation als hochgradig flexibel. 'Defekte Teile' werden problemlos während des laufenden Betriebs der Maschine ersetzt, die Gesamtleistung des Systems nimmt dabei nur unwesentlich ab. Und in der Tat sind auch heute die Effizienz- und Leistungsprobleme, die in großen Firmen oder Organisationen auftreten, strukturell gleichzusetzen mit den Problemen in komplexen Computersystemen. Bei einer derartigen Betrachtungsweise wird klar, dass in der Megamaschine wie auch im modernen Computer oder der Organisationsstruktur heutiger Konzerne Grundmechanismen am Werke sind, die weit über den Bereich der Technik oder der modernen Zivilisation hinausreichen. Es ist vielmehr zu vermuten, dass die Prinzipien, die Mumfords Idee der Megamaschine zugrunde liegen, noch sehr viel weiter zurückreichen und seit Anbeginn der Menschheit existieren. Wenn wir diese Strukturen nicht mehr den Objekten zuschreiben, sondern sie im denkenden Individuum selbst verorten, wird klar, dass Maschinen, wirtschaftliche Organisationsstrukturen oder Erklärungsmodelle der Naturwissenschaften lediglich einzelne Ausprägungen desselben Grundprinzips sind.

9 Ebd., S. 220.

10 Siehe weiter unten zum Thema Hierarchie, sowie Zwischenspiel: Komplexität – Abstraktionen, Module und Hierarchien.

1.2.2 Automaten und mechanische Rechenmaschinen

„Die Maschine beunruhigt uns auf ähnliche Weise wie die Idee des Gespenstes: etwas Lebloses bewegt sich und lebt, das heißt: es simuliert Leben. Hebt man die mechanischen Bewegungen der Maschine gegen die Regungen des organischen Lebens ab, so läuft das nicht darauf hinaus, dass die Maschine zum Todessymbol wird. Nicht auf den Tod weist sie hin, sondern auf die Möglichkeit, dass unser eigenes Leben, wie das ihre, ein bloß simuliertes sein könnte."

Lars Gustafsson[11]

Die ersten Maschinen im klassischen Sinn, d.h. mechanische Vorrichtungen aus miteinander verbundenen beweglichen Teilen, sind vor allem durch die Historien Herodots (ca. 490 – 430 v. Chr.) überliefert. In neun Büchern beschreibt er technologische Errungenschaften aus der assyrischen, ägyptischen, hellenischen und persischen Kulturgeschichte. In seinen Darlegungen beleuchtet Herodot nicht nur die sozialen Beziehungen, die Sitten und Gewohnheiten, die Zivilisationsmerkmale und die alltäglichen Verrichtungen der einfachen Menschen, sondern auch den jeweiligen Stand der Technik.[12] Neben Beschreibungen der größten Bauwerke der Antike finden sich auch einfache technische Ideen und Erfindungen des 5. Jahrhunderts vor Christus. In seinem zweiten Buch erwähnt Herodot bewegliche Statuen (Neurospasta) die mit verschiedenen Gelenken versehen sind. Damit der schakalartige Gott Anubis beim ägyptischen Totenkult möglichst lebendig und unsterblich erscheint, wurde er mit beweglichen Gelenken an Armen, Beinen und am Kiefer ausgestattet. Herodot berichtet an dieser Stelle auch von kleinen Statuen mit beweglichem Glied. Er schreibt:[13]„Sonst aber feiern die Ägypter das Dionysosfest fast genau so wie die Hellenen, nur ohne Chöre. Statt der Phallen aber haben sie sich was anderes ausgedacht, Figuren etwa eine Elle (48 cm) groß, mit einer Schnur, um daran zu ziehen, und die tragen die Frauen in den Dörfern umher, und das Glied hebt und senkt sich, und es ist nur wenig kürzer als die ganze Gestalt."

Wie Herodot beschreibt, wurden hier die Bewegungen noch von Menschenhand, zum Beispiel mit Hilfe von Schnüren, ausgeführt. Die ersten wirklichen Automaten, d.h. Selbstbeweger, wurden aber vermutlich etwa um die gleiche Zeit entwickelt. So enthält das Werk *Pneumatica* des Griechen Philon von Byzanz bereits verschiedene Theorien über das Wesen der Luft und des Vakuums, über Springbrunnen und andere hydraulische, mit Wind, Wasser und Dampf betriebene Geräte.[14] Einen ersten Höhepunkt erreichte die Technologie im ersten Jahrhundert n. Chr. mit den Automaten des antiken Mathematikers

11 In: Die Maschinen, in: Utopien, S. 36.

12 Vgl. Dimitri Kalligeropoulos, Technologische Funde aus den Historien Herodots.

13 Ebd., S. 62 f.

14 Vgl. Annette Beyer, Faszinierende Welt der Automaten.

und Ingenieurs Heron von Alexandria. Seine hydraulisch-pneumatischen Tempelautomaten und Spielwerke waren bis in die Renaissance von Bedeutung, als schließlich die Getriebe- und Räderwerktechnik die führende Rolle übernahm. Mittlerweile wurden verschiedene Automaten – z.B. der Äolsball als primitiver Vorläufer einer Dampfmaschine, die automatischen Tempeltüren oder der Vogelbrunnen – wie sie in den Sammelwerken Herons zur Pneumatik und im Automatentheater beschrieben sind, nachgebildet. Damit konnte die Funktionstüchtigkeit dieser technischen Ideen nachgewiesen werden.[15] Bei Heron von Alexandria findet sich auch die erste programmierbare mechanische Vorrichtung. Es wird ein fahrender Automat beschrieben, der seine Antriebskraft entweder durch eine Art Feder aus Sehnen erhält, ähnlich dem Prinzip eines Katapults, oder durch ein hängendes Bleigewicht. Die Bewegung des fallenden Gewichtes wird über ein Seil auf die Antriebsachse des Automaten geleitet. Die Fallgeschwindigkeit des Gewichtes, das sich in einem sanduhrartigen Kasten befindet, wird durch die Ausströmgeschwindigkeit der feinen Körner gesteuert. Die Programmierung der Bewegung wurde durch entsprechende Schnurwicklung auf der Drehachse bewerkstelligt. Heron schreibt im Automatentheater hierzu:[16]„Man stellt Tempel oder Altäre von mäßigem Umfange her, die sich von selbst heranbewegen und an bestimmten Punkten halten. Dann bewegt sich jede von den darin befindlichen Figuren entsprechend dem vorliegenden Plane oder einer passenden Fabel für sich und kehrt schließlich nach ihrem ursprünglichen Platze zurück.“

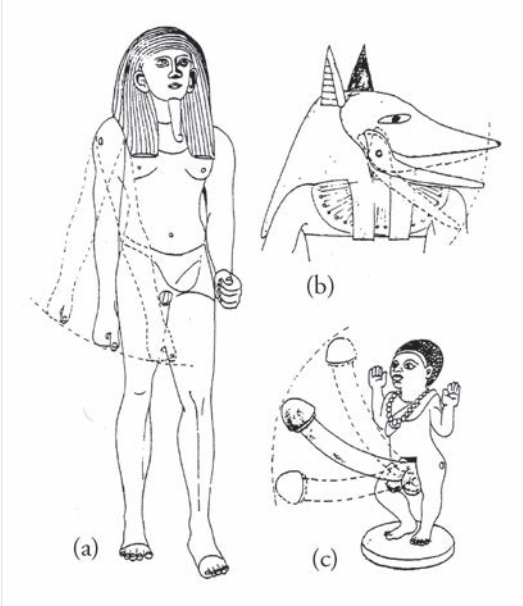


Abbildung 7:
Bewegliche ägyptische Statuen. (a) Beweglicher Arm, (b) Beweglicher Kiefer des Gottes Anubis, (c) Dionysos Statue mit beweglichem Glied. Solche Abbildungen sind nachgestellt und stützen sich auf die technischen Beschreibungen Herodots.

Automatenbauer arbeiten und wirken das gesamte Mittelalter hindurch, und wenn sich phasenweise kein Automatenbauer finden lässt, so sind doch die Fiktion und die Literatur mit Automaten gefüllt, die angeblich Phantastisches zu leisten vermochten. Ihren Höhepunkt erreicht die mechanische Linie des Automatenbaus im 18. Jahrhundert mit der berühmten Ente Vaucansons und den

15 Vgl. Dimitri Kalligeropoulos, Funktionelle Modelle aus den Automaten Herons.
16 Ebd., S. 75.

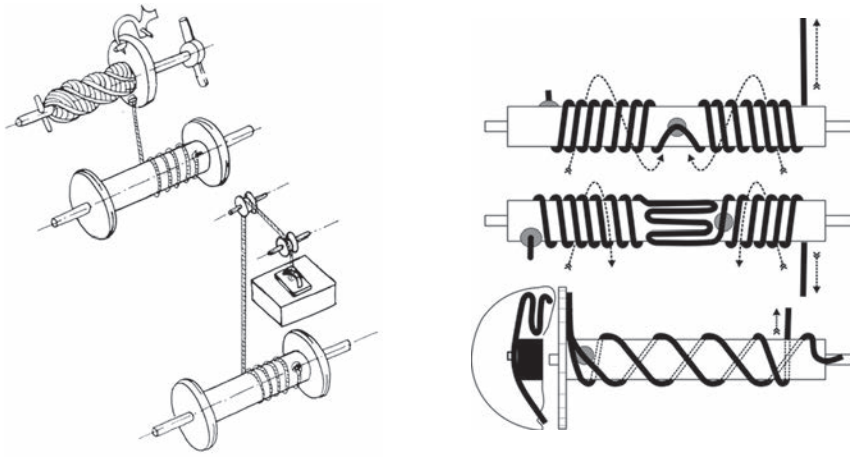


Abbildung 8/9:
Antriebsmechanismen des Bacchus-Tempelchens von Heron von Alexandria (links). Programmierung der Bewegung des Automaten durch geeignete Schnurwicklung. Linkswicklung bedeutet Vorwärtsbewegung, Rechtswicklung Rückwärtsbewegung und freie Schnurwicklung bedeutet Stillstand (rechts).

programmierbaren Androiden der Schweizer Automatenbauerfamilie Jaquet-Droz. Die feinmechanische Perfektion muss für die damalige Zeit als wahres Wunder gegolten haben und bis heute finden diese Arbeiten unsere Anerkennung. Die drei berühmten Automaten – der Zeichner, der Schriftsteller und die Musikerin – von Pierre Jaquet-Droz, seinem Sohn Henri-Louis, sowie Jean Frédéric Leschot und einigen weiteren Handwerkern, wurden im Jahre 1774 im Salon der Familie in La Chaux-de-Fonds erstmals der Öffentlichkeit vorgestellt. Das in unserem Zusammenhang Interessante an den Automaten ist ihre Programmierbarkeit. Sätze auswechselbarer Nocken erlauben die Ausführung unterschiedlicher Tätigkeiten. So kann der Zeichner bereits vier verschiedene Zeichnungen ausführen. Mit René Descartes gewannen Automaten – lange vorher – auch philosophisch an Bedeutung. Im *Discours de la méthode* (1637) und im *Traité de l'homme* (1677) setzt er Tiere mit Maschinen gleich. La Mettrie erweitert das Prinzip in *L`homme Machine* (1748) auf den Menschen.

Die beweglichen Statuen und Automaten hatten keine Funktion außer der Bewegung selbst, die bis ins 18. Jahrhundert als Mysterium galt und die Grenze zwischen Lebendigem und Leblosem markierte. Bereits im 17. Jahrhundert wurden mechanische Fertigkeiten aber auch für andere Ziele, nämlich den Bau von Rechenmaschinen eingesetzt. Unter Rechenmaschinen verstand man in der weitläufigsten Bedeutung mechanische Werkzeuge, die durch Bewegung und Kombination ihrer Teile gewisse arithmetische Operationen ausführen konnten. Während in Europa der 30-jährige Krieg tobte, entwickelte sich die Naturwissenschaft im Einklang mit der Mathematik zur führenden geistigen Macht. Kennzeichnend für diese Zeitspanne war der allmähliche Durchbruch zur mathematischen Behandlung naturwissenschaftlicher Phänomene. Mit der

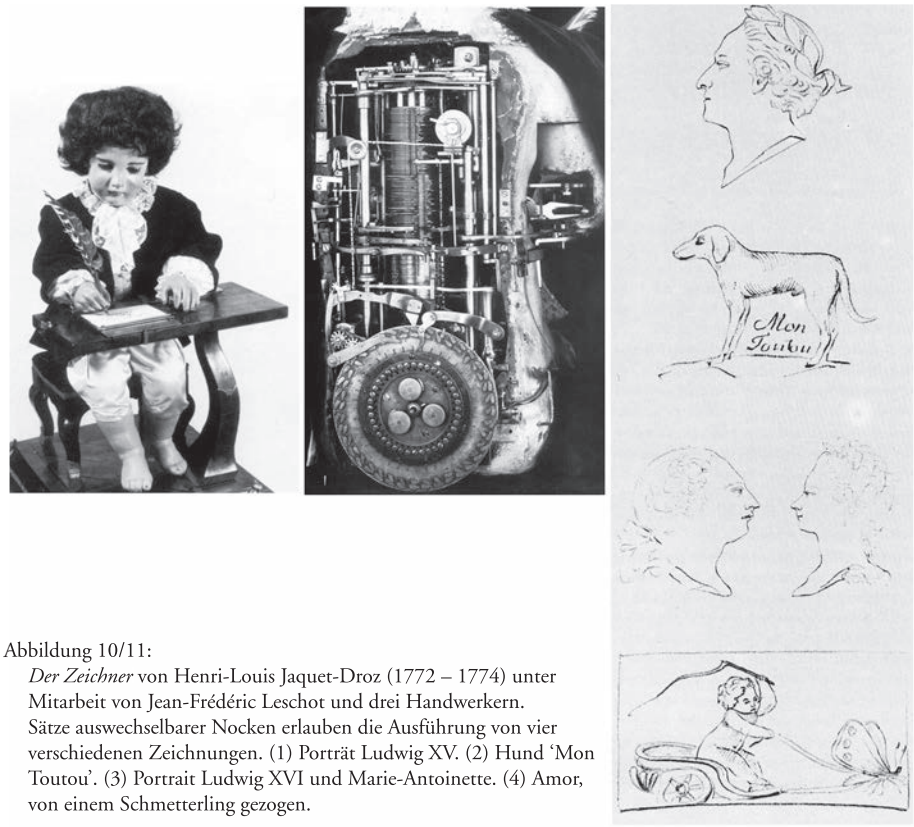


Abbildung 10/11:
Der Zeichner von Henri-Louis Jaquet-Droz (1772 – 1774) unter Mitarbeit von Jean-Frédéric Leschot und drei Handwerkern. Sätze auswechselbarer Nocken erlauben die Ausführung von vier verschiedenen Zeichnungen. (1) Porträt Ludwig XV. (2) Hund 'Mon Toutou'. (3) Portrait Ludwig XVI und Marie-Antoinette. (4) Amor, von einem Schmetterling gezogen.

Ausbreitung der neuen Wissenschaften wuchs insbesondere auch in der Astronomie der Bedarf an numerischen Berechnungen. Es ist deshalb nicht überraschend, dass in diese Zeit auch die ersten Experimente mit Rechenmaschinen fallen. Interessant ist in diesem Zusammenhang aber die Feststellung, dass die ersten mechanischen Rechner von Philosophen und Theologen entworfen wurden. Diese Epoche war durchdrungen von der Idee, geistige Leistungen, in deren Zentrum die Mathematik und das Rechnen standen, zu mechanisieren. Mit den indischen Zahlen[17] waren auch die praktischen Voraussetzungen für die Mechanisierung des Rechnens gegeben. Dieses Zahlensystem, das mit der linearen Ziffernreihe und dem Zehnerübertrag in sich selbst schon eine Art Urmaschine bildet, ermöglicht es, Rechnen als schematische Umformung von bedeutungslosen Zeichen durchzuführen. Uns sind heute drei Rechenmaschinen aus dem 17. Jahrhundert überliefert, die von den bedeutendsten Philosophen und Wissenschaftern dieser Zeit entwickelt wurden. Die älteste heute bekannte Rechenmaschine stammt vom Tübinger Professor für biblische Sprachen, Wilhelm Schickard (1592 – 1635). Schickard war nicht nur Sprachgelehrter, sondern auch Astronom, Geograph und

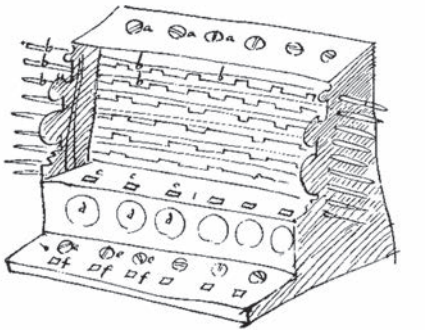
17 Siehe Zwischenspiel: Von den indischen Zahlen zur Leibnizschen Dyadik.

Mathematiker, was für diese Zeit, in der es noch die Spezies des Universalgelehrten gab, allerdings nichts ungewöhnliches war. Im Nachlass Schickards finden sich zwei Briefe an den Astronomen Kepler, sowie zwei kleine Federzeichnungen, die den Aufbau der Maschine zeigen, dazu ein Notizzettel mit Anweisungen für den Mechaniker, der die Maschine bauen sollte. Am 20. September 1623 schreibt Schickard an Kepler:[18] „... Dasselbe, was Du rechnerisch gemacht hast, habe ich in letzter Zeit auf mechanischem Wege versucht, und eine aus elf vollständigen und sechs verstümmelten Rädchen bestehende Maschine konstruiert, welche gegebene Zahlen augenblicklich automatisch zusammenrechnet: addiert, subtrahiert, multipliziert und dividiert. Du würdest hell auflachen, wenn Du zuschauen könntest, wie sie die Stellen links, wenn es über einen Zehner oder Hunderter weggeht, ganz von selbst erhöht bzw. beim Subtrahieren ihnen etwas wegnimmt." Am 25. Februar 1624 schreibt Schickard einen zweiten Brief an Kepler:[19] „Von dem mathematischen Gerät werde ich ein ander Mal eine genauere Abbildung geben; für heute nimm in Eile diese: aaa sind die Köpfchen aufrechter Zylinder, denen die Multiplikationen der Fingerzahlen aufgeschrieben sind, und sie schauen, soweit man ihrer bedarf, durch die ziehbaren Fensterchen bbb heraus. ddd haben innen fest angemachte Rädchen mit zehn Zähnen, die so ineinander greifen, dass, wenn irgendein rechts stehendes zehnmal gedreht wird, das links anschließende einmal herumgeht, oder, wenn jenes hundertmal herumgeht, das dritte einmal vorwärts bewegt wird usw., und zwar nach derselben Richtung, was die Einführung eines ähnlichen Rädchens erforderlich machte. ... Die jeweilige Zahl wird in den Löchern ccc auf dem mittleren Brett sichtbar. Schließlich deutet e auf dem untersten Brett Wirbel und f in ähnlicher Weise Löcher zum Sichtbarmachen der Zahlen an, deren man während der Operation bedarf. Aber das kann man so hastig nicht schreiben: leichter ist es am Objekt zu verstehen. Nun hatte ich für Dich bei dem hier ansässigen Johann Pfister ein Exemplar in Auftrag gegeben, dieses ist jedoch halbfertig zusammen mit anderen Sachen von mir, vor allem etlichen Kupferplatten, vor drei Tagen einer Feuersbrunst zum Opfer gefallen, die in der Nacht unversehens dort ausgebrochen ist. Darüber wird Dir Mütschelin ausführlicher berichten können. Den Verlust nehme ich schwer, jetzt zumal, wo er keine Zeit hat, rasch Ersatz zu schaffen." Schickards Leben war geprägt von den Unruhen des Dreißigjährigen Krieges. Zweimal flieht er aus Tübingen, 1634 bricht dort die Pest aus, der auch Schickard am 23. Oktober 1635 zum Opfer fällt.

Lange Zeit galt der Franzose Blaise Pascal (1623 – 1662), der wahrscheinlich von der öffentlichen Zurückhaltung Schickards profitierte, als der erste Erfinder einer Rechenmaschine. Die Wiederentdeckung der Schickardschen Maschine ist dem Keplerforscher Franz Hammer zu verdanken, der seine Funde im Jahre 1957 auf einem Kongress von Mathematikhistorikern vorstellte. Der junge Physiker

18 Siehe Zwischenspiel: Von den indischen Zahlen zur Leibnizschen Dyadik.

19 Bruno Baron von Freytag-Löringhoff, Wilhelm Schickards Tübinger Rechenmaschine von 1623, S. 7 f.



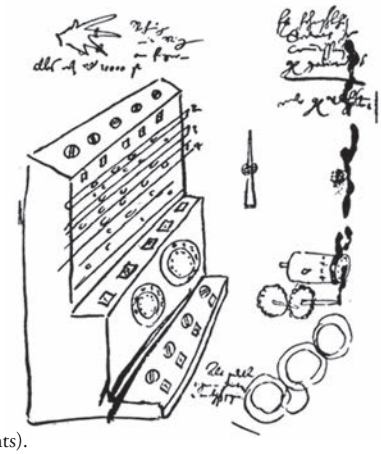


Abbildung 12/13:
Skizze zum zweiten Brief Schickards an Kepler (oben). Die Zeichnung der Rechenmaschine stammt aus dem Nachlass Keplers. Skizze aus dem Nachlass Schickards (rechts).

und Mathematiker Pascal war offensichtlich geschäftstüchtiger als Schickard und führte seine Maschine öffentlich vor. Er erreichte schon in sehr jungem Alter Anerkennung in Physik und Mathematik. Pascal formulierte unter anderem die Grundlagen der Wahrscheinlichkeitsrechnung und Statistik sowie das nach ihm benannte Pascalsche Gesetz, nach dem sich in Flüssigkeiten der Druck gleichmäßig in alle Richtungen ausbreitet. Um ca. 1643 entwickelt er seine Zwei-Spezies-Maschine – die so genannte Pascaline – zur Addition und Subtraktion von Zahlen. Wir hatten oben angedeutet, dass die ersten Rechenmaschinen nicht aus den Bedürfnissen der Wirtschaft und des Handels entstanden sind und weniger wegen ihres praktischen Nutzens entwickelt wurden. Für Pascals Rechenmaschine scheint das den Überlieferungen zufolge nicht zutreffend zu sein. Sein Vater war Steuerkommissar und die Maschine sollte ihm helfen, seine umfangreichen Berechnungen durchzuführen. Interessant an Pascals Maschine ist die Umschaltung zwischen den beiden Rechenarten Addition und Subtraktion. Durch verschieben der Abdeckplatte, auf der die jeweils komplementären Zahlen zur Zahl Neun angegeben waren, konnte auf sehr elegante Weise zwischen den Grundrechenarten gewechselt werden. Das Verfahren macht sich das mathematische Prinzip zunutze, dass die Subtraktion einer Zahl auf die Addition ihres Komplements zurückgeführt werden kann. Zum Beispiel gilt: 647 – 583 = 64. Das Ergebnis kann auch erreicht werden, indem man von der zu subtrahierenden Zahl das Komplement (die Ergänzung zur Zahl 999, bei 583 ist dies die Zahl 416) bildet, und diese addiert. Also 647 – 583 = 647 + 416 = 1064. Die führende Eins entfällt bei der mechanischen Realisierung, da sie durch kein Zahlrad repräsentiert ist.

Die dritte bedeutende Rechenmaschine des 17. Jahrhunderts stammte von dem Universalgelehrten Gottfried Wilhelm Leibniz (1646 – 1716). Er nannte seine Maschine eine ‘lebendige Rechenbank’, weil alle Zahlen ‘sich selbst rechnen’, d.h. addieren, subtrahieren multiplizieren und dividieren. Seine Vier-Spezies-Maschine ist vor allem für die Verwendung des Prinzips der Staffelwalze bekannt.

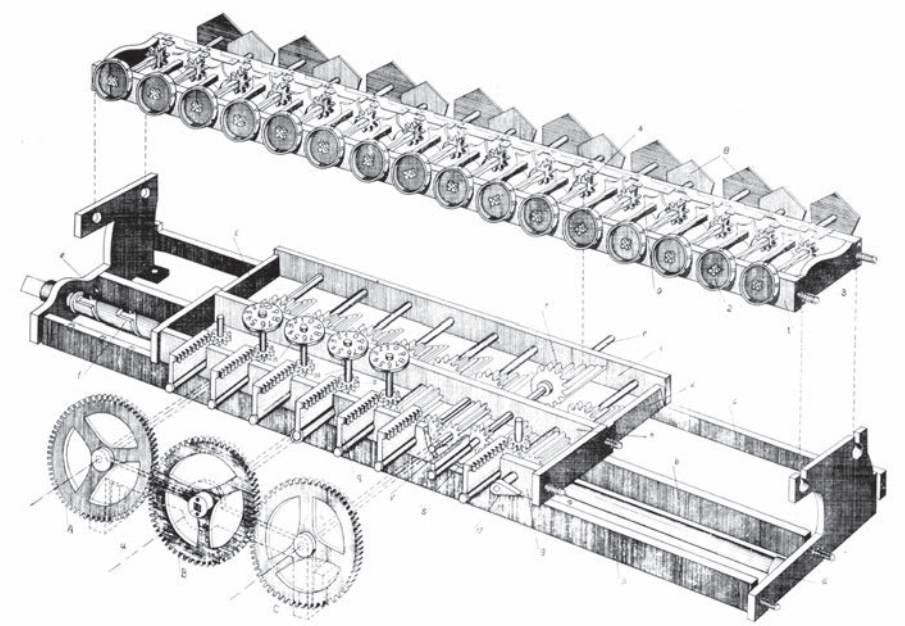
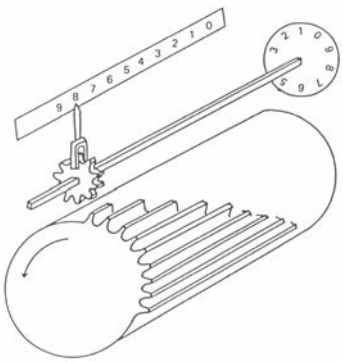


Abbildung 14:
Die Leibnizsche Rechenmaschine.

Dieses von Leibniz erdachte Prinzip zur Realisierung des Zehnerübertrags wird 50 Jahre später auch vom Pfarrer und Ingenieur Philipp Matthäus Hahn (1739 – 1790) wieder verwendet werden. Die Lösung des Problems des Zehnerübertrags war die größte mechanische Herausforderung der Rechenmaschinen. Mit dem Überlauf einer Stelle über die 9 hinaus, musste die Walze der nächst höheren Stelle mitbewegt werden. Soll etwa auf die Zahl 9999 eine 1 addiert werden, dann müssen die Walzen nach der Berechnung die Zahl 10000 zeigen, d.h. es



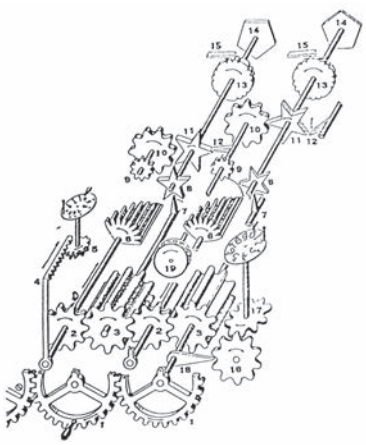


Abbildung 15/16:
Das Prinzip der Staffelwalze in der Leibnizschen Rechenmaschine.

müssen zuverlässig und reibungslos in vier Stellen gleichzeitig Überträge erfolgen. Dies war für die damalige Zeit eine mechanische Herausforderung. Leibniz löst das Problem mit seiner Staffelwalze, Pascal durch einen Hebelmechanismus (siehe Abbildung 15 – 17) und Schickard durch fehlende und abgefeilte Zähne. Wir werden Leibniz später im Abschnitt Formalisierung noch einmal aufgreifen, weil bei ihm die Gedanken zu formalen Logik und die algorithmische Orientierung seiner gesamten Wissenschaft am weitesten ausgearbeitet sind. Er verkörpert von den hier behandelten Personen am stärksten das neue Weltbild der Naturwissenschaften. Den Computerwissenschaften ist er nicht nur wegen seiner Rechenmaschine bekannt, sondern gleichfalls als Erfinder des Dualsystems und vielen gilt er sogar als Begründer der künstlichen Intelligenz. Außerdem gilt er neben Newton als Begründer der Differentialrechnung. Am Bedeutendsten in unserem Zusammenhang sind aber sicher seine Beiträge zur Formalisierung der Logik und Algorithmisierung mathematischer Problemlösungsverfahren.

Zu den vorgestellten Rechenmaschinen ist abschließend festzuhalten, dass sie nicht programmierbar sind. Wir können deshalb bei diesen Maschinen auch nicht von Computern sprechen, die sich bekanntlich über ihre Programmierbarkeit definieren. Es gibt in den ersten mechanischen Rechenmaschinen keinen eigenen Bereich für die Spezifizierung von Operationen, vielmehr ist der Mechanismus für die Ausführung der Operationen die Konstruktion der Maschine selbst. Heute würde man sagen, die Operationen sind ‘festverdrahtet’. Erst wenn die Maschine einen Bereich vorsieht, in den die eigentliche Funktion der Maschine noch eingeschrieben werden kann, können wir von Programmierbarkeit sprechen. Beim fahrenden Automaten war das Programm die Art der Seilwicklung, bei den Automaten der Jaquet-Droz’ die Struktur der auswechselbaren Nocken.

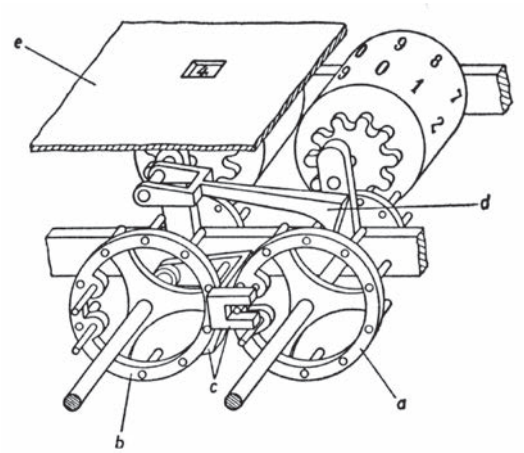


Abbildung 17:
Darstellung der Pascalschen Rechenmaschine (Pascaline) für zwei Stellen. Durch verschieben der Abdeckplatte kann zwischen Addition und Subtraktion umgeschaltet werden.

1.2.3 Die rationalistische Tradition

„Überhaupt verhält es sich mit allen solchen Kalküls so, dass man durch sie nichts leisten kann, was nicht auch ohne sie zu leisten wäre; der Vorteil ist aber der, dass wenn ein solcher Kalkül dem innersten Wesen vielfach vorkommender Bedürfnisse korrespondiert, jeder ... die dahin gehörigen Aufgaben lösen, ja selbst in so verwickelten Fällen gleichsam mechanisch lösen kann, wo ohne eine solche Hilfe auch das Genie ohnmächtig wird. So ist es mit der Erfindung der Buchstabenrechnung überhaupt, so mit der Differentialrechnung gewesen, ...“
Carl F. Gauß[20]

Nach Mumford findet im Spätmittelalter die ‘Wiederkehr des Sonnengottes’ statt, die durch das heliozentrische Weltbild Keplers und anderer Wissenschaftler eingeleitet wird. Das neue Denken, das große technische Transformationen mit sich brachte, hatte sich ab dem sechzehnten Jahrhundert durchgesetzt. Neben Kepler sind Kopernikus, Galilei, Descartes, Leibniz und Newton zentrale Figuren des neuen wissenschaftlichen Vorgehens. Mit den neuen Wissenschaften, die nur das untersuchen und gelten lassen, was objektiv messbar ist, setzt sich immer mehr ein mechanistisches Weltbild durch.[21] Alles was in dieser neuen physikalischen Welt noch existiert, sind Massen und Bewegungen in einem abstrakten Raum und einer ebenso abstrakten Zeit. Alles Lebendige wird im Interesse der Objektivität eliminiert. In dieser Welt wird radikal getrennt zwischen Innerem und Äußerem, zwischen Subjekt und Objekt, zwischen Quantität und Qualität. Diese Wissenschaft, die selbst den Menschen als Automaten begreift, wird im Laufe der Zeit immer mehr zur neuen Religion, und ihr Versprechen ist der unbegrenzte Fortschritt. Die neue Megamaschine setzt sich nicht mehr nur aus menschlichen Komponenten zusammen, sondern sie wird ergänzt durch mechanische und inzwischen auch elektronische und chemische Submaschinen. Eine neue wichtige Grundkomponente der Megamaschine ist der Kapitalismus, der den Profit ins Zentrum des Handels stellt. Die Menschen werden nicht mehr durch Religion oder Gewaltandrohung in der Megamaschine gehalten, sondern durch die profane Notwendigkeit, den eigenen Lebensunterhalt zu verdienen. Entzauberung, Rationalität und exzessive Ausdifferenzierung sind Begriffe, die mit der modernen Wissenschaft und den mit ihr einhergehenden Lebensbedingungen verbunden sind. Modern sein heißt vor allem, im Jetzt zu leben und aus dieser Gegenwart und der eigenen Verantwortung das individuelle Denken und Handeln abzuleiten.

Mumford nennt vor allem Galileo Galilei (1564 – 1632) als einen der bedeutenden Geburtshelfer des neuen Weltbildes. Er vereinigt als erster empirisches (unmittelbare Beobachtung) und theoretisches Wissen (Abstraktion, Mathematik, Formalismen) und formt daraus ein mächtiges Werkzeug. Die

20 In einem Brief an Schuhmacher von 15.5.1843, zitiert nach Niklaus Wirth, Systematisches Programmieren, S. 5.

21 Siehe auch Zwischenspiel: Die Weltmaschine.

Verbindung aus analytischer und synthetischer Methode ist das Kernstück der neuen Wissenschaft. Ausgangspunkt ist der menschliche Verstand, der das Organ des Denkens ist und weiter dringt als die Sinneswahrnehmung. Da die Sinneswahrnehmung allein trügen kann, muss der Verstand den Mangel an Erfahrung ergänzen und die Realität aller Erscheinungen feststellen. Das höchste Ziel der Philosophie muss daher die Einheit des Gesichtspunktes und damit die mathematische Durchdringung aller Erfahrungstatsachen sein. Überall seien die verwickelten Erscheinungen der sinnlichen Beobachtungen über die 'metodo risolutivo' in ihre einfachsten Komponenten zu zerlegen, um dann aus ihnen die Vorgänge durch die 'metodo compositivo', die synthetische Methode, zu erklären. Quantitäten spielen bei diesem Vorgang eine besondere Rolle, Erscheinungen werden auf Quantitäten und deren mathematische Beschreibungen zurückgeführt. Galilei selbst drückt dies so aus:[22] „Die Philosophie steht in dem großen Buch, dem Universum, das unserer Betrachtung stets offen steht. Aber man kann das Buch nicht verstehen, wenn man nicht zuerst lernt, die Sprache zu begreifen und die Buchstaben zu lesen, aus denen sie zusammengesetzt ist. Es ist geschrieben in der Sprache der Mathematik, und seine Buchstaben sind Dreiecke, Kreise und andere Figuren, ohne die es unmöglich ist, auch nur ein einziges Wort davon zu verstehen; ansonsten irrt man in einem dunkeln Labyrinth umher."

Mit Galilei wird erstmals der Dualismus zwischen objektiver und subjektiver Welt eingeführt. Seine Methode, die inzwischen durch die Wissenschaften perfektioniert, ausdifferenziert und verfeinert wurde, ist heute unter dem Begriff der 'rationalistischen Tradition' bekannt. Hinter dem Begriff Rationalismus steht die Idee, alleine durch den Einsatz von Vernunft zu Erkenntnissen zu gelangen. Der amerikanische Philosoph John R. Searle fasst die Grundprinzipien der westlichen rationalistischen Tradition aus heutiger Sicht wie folgt zusammen:[23]

1. Realität existiert unabhängig von ihren Darstellungen durch Menschen.
2. Zumindest eine der Funktionen von Sprache besteht darin, Bedeutungen von Sprechern zu Hörern zu übermitteln, und manchmal ermöglichen diese Bedeutungen es der Kommunikation, sich auf Gegenstände und Zustände in der Welt zu beziehen, die sprachunabhängig existieren.
3. Wahrheit ist eine Frage der Genauigkeit der Darstellung.
4. Wahrheit ist objektiv. Die Welt besteht gewissermaßen aus einer Sammlung von interpretationsunabhängigen Fakten.

Ein wichtiger Schlüssel zum Verständnis der rationalistischen Tradition und der Naturwissenschaften generell ist das Experiment. Galilei fragt nicht mehr, warum die Dinge fallen, sondern wie sie fallen. Erst im Experiment tritt der Mensch aus seiner Passivität gegenüber der Natur heraus und entwickelt ein neues aktives Verhältnis. Es geht nun darum, der Natur durch 'Provokation' ihre mathematischen

22 Zitiert nach Lewis Mumford, Mythos der Maschine, S. 395.

23 Vgl. John Searle, Rationalität und Realismus oder Was auf dem Spiel steht, S. 377 – 391.

Gesetzmäßigkeiten zu entlocken. Hinter der Vielfalt der Erscheinungen werden wenige einfache Grundgesetze vermutet, möglicherweise nur ein Einziges, die so genannte Weltformel. Über diese vermuteten Gesetzmäßigkeiten werden im ersten Schritt Hypothesen gebildet. Das Experiment wird dann zum Prüfstein der Wahrheit, indem die Natur gezwungen wird, Antworten auf die Hypothesen zu geben. Obwohl heute die meisten Wissenschaftler zustimmen würden, dass die auf diese Weise gefundenen Gesetzmäßigkeiten nur im Kopf des Menschen und in Form formaler Beschreibungen existieren, ist das Bild der Objektivität noch stark in den Naturwissenschaften verwurzelt. Die Bedeutung Galileis liegt darin, dass er einen Weg öffnet für eine neue physikalische Weltbetrachtung. Heute sehen wir den Wahrheitsgehalt der Physik nicht mehr in einer objektiven Spiegelung der Realität in den Theorien, sondern in strukturerhaltenden Beziehungen. Da diese Strukturverwandtschaften (Homomorphien und Isomorphien) keine objektgetreue Abbildung mehr voraussetzen, darf in den Wissenschaften auch mit bloßen Symbolen gearbeitet werden, denen kein Element der äußeren Realität mehr entspricht. Die Beschreibung der Realität als formales System funktionaler Abhängigkeiten ist auch als 'Entmaterialisierung des Weltbildes' bekannt – Substanz wird hier durch Struktur ersetzt.

Neben Galileo Galilei gilt René Descartes (1596 – 1650) als weitere Leitfigur der rationalistischen Tradition. In seinem kleinen Buch *Le Discours de la Méthode* legt er die vier Grundsätze rationalen Denkens nieder.[24] Die Form des Denkens, die bei Descartes formuliert wird, zieht sich in der Folge durch die gesamte abendländische Philosophie und findet in der Kybernetik einen vorläufigen Höhepunkt. In moderner Fassung lauten die Handlungsanweisungen für die moderne Wissenschaft wie folgt:[25]

1. Beobachtung eines Phänomens, das als zu erklärendes Problem angesehen wird.
2. Entwicklung einer erklärenden Hypothese in Form eines deterministischen Systems, das ein Phänomen erzeugen kann, welches mit dem beobachteten Phänomen isomorph ist.
3. Generierung eines Zustandes oder Prozesses des Systems, der entsprechend der vorgelegten Hypothese als vorhergesagtes Phänomen beobachtet werden soll.
4. Beobachtung des so vorhergesagten Phänomens.

Diese Abfolge elementarer Schritte muss zur Verfeinerung der wissenschaftlichen Aussage wiederholt durchlaufen werden. Algorithmische Problemlösungen und Computerprogramme stehen in direkter Tradition des rationalistischen Denkens. Die rationale Vorgehensweise ist nicht nur in den Naturwissenschaften das leitende Paradigma sondern auch bei der Entwicklung künstlich hergestellter Dinge, d.h. technischer Artefakte, die heute in aller Regel

24 Vgl. René Descartes, Von der Methode des richtigen Vernunftgebrauchs und der wissenschaftlichen Forschung, S. 15 f.

25 Siehe Winograd, Flores, Erkenntnis, Maschinen, Verstehen, S. 38.

ebenfalls auf der Basis formaler Beschreibungen produziert werden. Und genauso gehen wir auch vor, wenn wir Computerprogramme schreiben, d.h. wir zerlegen die Welt in ein Zusammenspiel indentifizierbarer Objekte mit wohldefinierten Eigenschaften und wenden dann schrittweise allgemeingültige Regeln an, um die so beschriebene hermetische Welt in ihrem zeitlichen Verhalten fortzuschalten. Sofern es sich nicht um naturwissenschaftliche Anwendungen, sondern um Entwurfsprozesse handelt, werden die Phänomene nicht aus der Beobachtung gewonnen, sondern als zweckorientierte Funktionen gesetzt.

1.2.4 Formalisierung

„Die zweiwertige Logik ist weder die Logik der Natur noch die des Subjekts. Sie ist allein die Logik des objektiven Subjekts, des gedachten Subjekts und des objektivierten Subjekts. Sie selbst produziert das, was sie beschreibt. Mit der transklassischen Maschine hat die reale Objektivierung des Subjektes einen neuen Höhepunkt erreicht. Eine Beschränkung auf die zweiwertige Logik macht die Menschen unvermeidlich zu Marionetten objektivierter Subjektivität, da das 'subjektive Subjekt' sich dieser Logik entzieht."

Eggert Holling, Peter Kempin[26]

Die Grundidee der Programmierung besteht in der Idee des operativen Symbolgebrauchs, d.h. Zeichen werden zu Befehlen, die sowohl von einem Menschen als auch von einer Maschine ausgeführt werden können. Die wichtigste Voraussetzung dafür ist zunächst die Formalisierung logischer Schlussweisen, die darin gründet, dass die Bedeutung von Zeichen vollständig von der regelbasierten Manipulation der Zeichen getrennt wird. In diesem Zusammenhang kehren wir noch einmal zurück zu G. W. Leibniz, der hier wiederum eine bedeutende Stellung einnimmt, da er den für seine Zeit radikalsten Versuch unternimmt, einen neuen Logik-Kalkül zu entwickeln. Natürlich waren in der Geschichte der Formalisierung über Jahrhunderte Vorarbeiten geleistet, bevor Leibniz diesen Schritt unternehmen konnte. Erwähnt seien hier nur stellvertretend die Rechenmeister und die maßgebliche Entwicklung der Buchstaben-Algebra durch Vieta.[27] Dennoch ist die Idee der Kalkülisierung der Logik im wesentlichen Leibniz zuzusprechen. Auch wenn seine Kalküle selbst nahezu ohne Einfluss blieben, hat er mit seinen Ideen den Boden für die Nachfolger bereitet. Zunächst zum Begriff des Kalküls. Ein Kalkül besteht ganz allgemein aus drei Elementen:

1. Einem Alphabet von Zeichen (Operanden),
2. aus grammatikalischen Regeln (Operatoren), die beschreiben, wie Zeichenketten in neue Zeichenketten umgeformt werden können, und
3. aus ausgezeichneten Zeichen oder Zeichenketten (den Axiomen), die als

26 In: Identität, Geist und Maschine, S. 138.

27 Siehe hierzu die Zwischenspiele: Algorithmisches Denken. Von den indischen Zahlen zur Leibnizschen Dyadik. Die Rechenmeister und die Cossisten.

Startkonfigurationen für Umformungsprozesse dienen.

Entscheidend an der Idee der Kalkülisierung ist, dass logische Operationen wie Rechenoperationen durchgeführt werden können und die verwendeten Zeichen sich nur noch innerhalb des Kalküls aufeinander beziehen. So gibt es Zeichen, die manipuliert werden (Operanden) und Zeichen die manipulieren (Operatoren). Wir werden dieser wichtigen Unterscheidung zwischen Operanden und Operatoren später wieder begegnen. Die Zeichen selbst besitzen dabei keinerlei Bedeutung mehr. Ihre Interpretation außerhalb des Kalküls bleibt auf der Stufe des operativen Gebrauchs vollkommen offen. Auf diese Weise wurde auch schon vor Leibniz durch die Einführung des indischen Ziffernsystems (mit dem bekannten Alphabet aus den zehn Ziffern und dem Mechanismus des Stellenwertes) die Kalkülisierung der Arithmetik möglich. So können wir seither zum Beispiel die Multiplikation lernen, ohne sie zu verstehen, ja ohne überhaupt zu wissen, dass wir rechnen. Diese Stufe der Operationalisierung der Arithmetik bildete die Voraussetzung für die vollständige Mechanisierung der Verfahren und die Bearbeitung durch Maschinen. Ganz allgemein können aus den Axiomen durch wiederholte Anwendung der Regeln alle gültigen Zeichenketten hergeleitet werden. Die Wahrheit von Aussagen und stringentem Denken wird so auf die Wohlgeformtheit von Zeichenketten abgebildet und 'bewiesen' wird durch die Kraft der Form. Auf der Basis von Kalkülen lassen sich damit sowohl mathematische als auch logische Beweise durchführen. Man kann sogar zeigen, dass jeder mathematische Beweis sich in einem formalen System als Folge von Elementaroperationen beschreiben lässt. Daraus folgt, dass Erkenntnisse durch maschinelles Anwenden von Regeln gewonnen werden können. Dies war die eigentliche Bedeutung und das wesentliche Ziel, das Leibniz mit seinem umfangreichen Forschungsansatz erreichen wollte. Diese Instrumentalisierung des Kalküls zum Zwecke philosophischer Erkenntnisgewinnung – heute als Leibnizprogramm bekannt – gliedert sich in drei miteinander in Beziehung stehende Ideen:[28]

1. Eine Universalwissenschaft: Die Idee einer 'scientia generalis' geht auf Raimundus Lullus (1232 – 1316) zurück, der mit seiner 'ars magna' eine universale Wissenschaft auf Basis der Kombinatorik gründen wollte. Die neue Logik, die hier emphatisch gesucht wurde, war vor allem darauf angelegt, bislang unbekannte Wahrheiten ans Licht zu bringen. Bei Leibniz teilt sich die 'scientia generalis' in die 'ars iudicandi' und die 'ars inveniendi'. Während die 'ars iudicandi' dazu dient, bekannte Aussagen auf ihre Wahrheit zu prüfen, soll die 'ars inveniendi' bisher unbekannte Aussagen generieren.
2. Eine universelle Kalkülsprache: Mit der 'characteristica universalis' strebt er eine neue Sprache an, die die Zweideutigkeiten und Fehlschlüsse unserer Alltagssprachen ausräumt. Zunächst muss hierzu eine Menge nicht weiter

28 Siehe auch Zwischenspiel: Logische Kalküle, Perzeptren und Neuronale Netze.

zerlegbarer Elementarbegriffe gefunden und diesen umkehrbar eindeutige Zeichen zugeordnet werden, die so genannten 'characteres'. Durch Synthese lassen sich aus diesen Grundbegriffen sämtliche Aussagen herleiten und durch Analyse lässt sich ein nicht-elementarer 'character' in seine Grundbegriffe zerlegen. Sind die 'characteres' sowie die Synthese- und Analysemethode richtig gewählt, dann werden sie unter sich auch in der gleichen Relation stehen, wie die Gedanken, die sie repräsentieren. Als Synthesemethode hatte Leibniz die Zahlenmultiplikation und als Analysemethode die Primfaktorenzerlegung von Zahlen vorgesehen.

3. Ein logischer Kalkül im engeren Sinne: Während die 'characteristica universalis' Ähnlichkeit mit symbolischen Logiken zeigt, kann sein Logik-Kalkül (calculus ratiocinator) als Vorläufer unserer heutigen mathematischen Logiken gelten. Das logische Schließen soll hier zu einem formalen Rechnen nach vorgegebenen Regeln werden.

Leibniz' Idee einer mathematischen Beschreibung der klassischen Logik gelingt erst dem Engländer George Boole (1815 – 1864) Mitte des 19. Jahrhunderts mit seiner 1854 veröffentlichten Arbeit *The Laws of Thought*. Mit seinem heute als Boolesche Algebra bekanntem Formalismus wird die gesamte abendländische Logik operationalisiert. Boole formuliert eine logische Algebra, die sich als restringierte Form der numerischen Algebra erweisen wird. Der Formalismus der Logik war aber nicht zuletzt gedacht als eine Methode, die Mathematik auf ein sicheres Fundament zu stellen, weniger als Substrat für die Verwirklichung technischer Anwendungen. Die logisch-formalen Methoden, die heute Grundlage der gesamten naturwissenschaftlichen Disziplinen und insbesondere aller Anwendungen der Informatik sind, wurden also nicht für deren Bedarf, sondern eigentlich für die Mathematik maßgeschneidert. Es bleibt also die Frage, ob eine technische Logik noch aussteht.

Wichtig ist, dass damit auch der Weg zur Realisierung der Aussagenlogik auf Hardwareebene offen stand.[29] Tatsächlich wird wenige Jahre nach bekannt werden der Booleschen Arbeit eine solche Maschine entwickelt. Im Jahre 1870 stellt W. S. Jevons der Royal Society in London seine Logik-Maschine vor, die wegen ihrer Tasten den Namen *Logisches Piano* erhält. Bei Eingabe einer Reihe von Prämissen liefert die Maschine den resultierenden Schlusssatz. Das Gebäude der Formalisierung hat mit Boole aber noch lange nicht seinen Abschluss gefunden, noch stehen die umwälzenden Ergebnisse des 20. Jahrhunderts aus. Noch einmal unternimmt ein bedeutender Wissenschaftler den Versuch, die Welt vollständig auf ein formales Fundament zu stellen. 1920 fordert der Göttinger Mathematiker David Hilbert (1862 – 1943), die gesamte Mathematik auf der Grundlage eines konsistenten Axiomsystems aufzubauen. Er war überzeugt, dass es möglich ist, die Mathematik so zu formalisieren, dass alle Aussagen innerhalb

29 Vgl. Sybille Krämer, Symbolische Maschine, S. 101.

eines einzigen Systems vollständig, widerspruchsfrei und entscheidbar dargestellt werden können. In einem Vortrag vor der deutschen Naturforschergesellschaft im Jahre 1922 begründet Hilbert seinen Plan:[30] „Meine Untersuchungen zur Neubegründung der Mathematik bezwecken nichts geringeres, als die allgemeinen Zweifel an der Sicherheit des mathematischen Schließens definitiv aus der Welt zu schaffen. Wie nötig eine solche Untersuchung ist, gewahren wir, wenn wir bedenken, wie wechselnd und unpräzise die diesbezüglichen Anschauungen oft selbst der hervorragendsten Mathematiker waren, oder wenn wir uns erinnern, daß von einigen der namhaftesten Mathematiker der neuesten Zeit die bisher für die sichersten gehaltenen Schlüsse in der Mathematik verworfen werden." Hilbert kann zu diesem Zeitpunkt noch nicht wissen, dass sein Programm an den grundlegenden Grenzen formaler Systeme scheitern wird. Der fünfundzwanzigjährige Österreicher Kurt Gödel zeigt im Jahre 1931 in einem Aufsatz mit dem Titel „Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme", dass es prinzipiell keinen Kalkül geben kann, innerhalb dessen alle mathematischen Beweise formuliert werden könnten. Der heute als 'Gödelscher Unvollständigkeitssatz' bekannte Beweis zeigt, dass dies selbst für den eingeschränkten Bereich der Arithmetik nicht möglich ist. In seinem Beweis konstruiert Gödel Aussagen über die natürlichen Zahlen, die aus den Axiomen der Arithmetik nicht bewiesen werden können, vorausgesetzt, diese ist widerspruchsfrei. Gödels Artikel ist wahrscheinlich der wichtigste Einzelbeitrag, der bis dato in der Geschichte der Mathematik und Logik abgeliefert wurde. Sein Beweis macht Hilberts Hoffnungen auf absolute Sicherheit in der Mathematik zunichte, indem er zeigt, dass das, was wir unter einem mathematischen Beweisverfahren verstehen, nicht mit der Anwendung einer axiomatischen Methode zusammenfällt. Rossers Version des Gödelschen Unvollständigkeitssatzes zeigt, dass bereits die Forderung der Widerspruchsfreiheit des Systems die Unvollständigkeit impliziert. Alonzo Church (1903 – 1995) und Alan M. Turing (1912 – 1954) weisen 1936 unabhängig voneinander nach, dass auch die Entscheidbarkeit formaler Systeme nicht gegeben ist. Das heißt, dass es keinen Algorithmus geben kann, der für jede mathematische Aussage beweist, ob sie wahr oder falsch ist. Gleichzeitig markiert Turings Publikation „On Computable Numbers, with an Application to the Entscheidungsproblem" den Beginn einer neuen Ära. In diesem Artikel formuliert Turing erstmals das formale Konzept einer universellen Maschine, die in der Lage ist, alle anderen Maschinen zu simulieren. Die Programmierbarkeit der Turing-Maschine stellt sicher, dass wir nicht unzählige Maschinen für unterschiedliche Aufgaben konstruieren müssen, sondern nur eine, die wir durch entsprechende Programmierung an alle Aufgaben anpassen können. Turings Arbeit bildet den (vorläufigen) Schlussstein einer über

30 David Hilbert, Die logischen Grundlagen der Mathematik, in: ders., Hilbertia, S. 33.

Jahrhunderte fortgeschriebenen Bemühung, unsere Maschinen einerseits immer leistungsfähiger zu machen und dabei andererseits unseren Maschinenbegriff immer weiter zu abstrahieren. Am Ende dieses Prozesses fallen im Computer Mechanisierung, Formalisierung und Logik in eins.

Holling und Kempin[31] weisen darauf hin, dass die Unterschätzung des Formalen eine lange Tradition hat, aber nicht nur in den Geisteswissenschaften, die Formalismen für ein begrenztes Hilfsmittel der Naturwissenschaften halten, sondern selbst in der Mathematik, die ihren Zeichensystemen keinerlei gesellschaftliche Bedeutung beimessen. Die eigentliche Bedeutung des Formalen als gesellschaftliche Integrationsinstanz und als implementierte Theorie, die von beiden Seiten übersehen wird, geht in der programmierbaren Maschine eine neuartige und mächtige Verbindung mit dem Mechanismus ein. Mit der Axiomatisierung und Formalisierung in der Mathematik und ihrer Anwendung in den Naturwissenschaften geht unvermeidlich die Verselbständigung der Zeichen einher. Bis Hilbert hatten die Zeichen noch einen – wenn auch bereits gelockerten – Bezug zur Welt gehabt. Die Postmoderne verzichtet dagegen vollständig auf Repräsentation. Die Zeichen bilden nun eine eigene Realität, die nichts mehr repräsentiert außer sich selbst. Die formalen Zeichen sind nur noch Platzhalter für beliebige Inhalte. Beziehungen zwischen den formalen Systemen und beobachtbaren Phänomenen beanspruchen keinerlei Allgemeingültigkeit oder gar 'Wahrheit' mehr, sondern unterliegen ausschließlich der Zweckmäßigkeit. Der Umgang mit dem von jeglicher Interpretation befreiten Formalismus beschränkt sich auf ein schematisches Operieren mit Zeichen. „Existiert der Formalismus erst einmal in der losgelösten Form, so ist es eine Frage der Pragmatik, ob wir ihn für angemessen halten, einen bestimmten Bereich der Realität für unsere Zwecke abzubilden. Für den Bereich der Realität mag die Frage von Belang sein, was wirklich 'wahr' ist oder 'falsch'. Der Formalismus bleibt davon völlig unberührt. Taucht das Problem auf, dass wichtige Bereiche der Realität durch das gewählte formale Modell nicht abgebildet werden können, so entsteht allenfalls die Frage, ob es das angemessene formale Modell ist, oder ob man nicht besser ein anderes sucht."[32] Diese nicht mehr an einen bestimmten Kontext gebundenen formalen Systeme sind darüber hinaus auch frei von Subjektivität. Formale Systeme (Kalküle) fußen in der abendländischen Rationalität und bieten damit eine gemeinsame Ebene, auf der sich die Mitglieder einer Gemeinschaft in all ihrer Verschiedenheit aufeinander beziehen können. Jedem denkenden Subjekt ist diese allgemeine Rationalität zugänglich. Solange diese Formen der Rationalität von der Gesellschaft allgemein anerkannt und angewandt werden, ist rationales Denken immer auch öffentliches, gesellschaftliches Denken. Alfred Sohn-Rethel geht sogar so weit, dass er die Erkenntnisprinzipien des Denkens

31 A.a.O., S. 81.

32 Ebd., S. 90.

– Kants 'Kategorien a priori' – nicht mehr dem Individuum zuordnet, sondern als geschichtlichen Prozess auffasst, der nur in bestimmten Gesellschaftsformen entsteht und diesen eigen ist. Erkenntnisfunktionen des Denkens wären somit allgemeine Organisationsprinzipien, die notwendig in warenproduzierenden, also geldvermittelten Gesellschaften als Folge der Tauschabstraktion auftreten. „Sie sind Erkenntnisprinzipien, die die begriffliche Grundlage der antiken Philosophie sowohl wie der modernen Naturwissenschaften bilden, und die wir der Einfachheit halber mit dem seit Kant geläufig gewordenen Namen der 'Kategorien a priori' etikettieren können. Diese Kategorien wären demnach Vergesellschaftungsformen des Denkens, die ein Individuum von genügender Intelligenz und Schulung befähigen, sich eines begrifflich unabhängigen Intellekts oder Verstandes zu bedienen, der für die Gesellschaft denkt. Dem Individuum selbst allerdings, zumal dem modernen, erscheint sein vergesellschaftetes Denken im Gegenteil als Leistung seines, dem Ursprung nach zwar mysteriösen (gottgegebenen und doch gottverneinenden), der Logik nach aber autonomen und ihm ureigenen 'ego cogitans'. Unserer Erklärung gemäß sind jedoch diese Kategorien gesellschaftlich vorgeformt und daher so, wie sie ans Individuum gelangen, ihm in fertiger (aber darum noch lange nicht unmittelbar identifizierbarer) Form gegeben, also tatsächlich Kategorien 'a priori' und auch für alle Individuen dieser Gesellschaft natürlich identisch dieselben." [33]

Voraussetzung für die Teilnahme am 'öffentlichen Denken' ist Abstraktionsfähigkeit, die gemeinhin als höchste Stufe der rationalen Entwicklung des Menschen angesehen wird. Da rationales Denken, wie wir gesehen haben, öffentlichen Charakter besitzt, ist es geeignet, eine allgemeine Bezugsebene zu bilden, die in der Lage ist, universelles Wissen auszubilden. So ist zum Beispiel wissenschaftliches Wissen immer öffentliches Wissen, das in codierter Form vorliegt und von jedem verstanden werden kann, der es versteht, den Code zu lesen. Die Codes beschreiben somit gesellschaftliche Erfahrungen, die jeder individuell nachvollziehen kann, indem er die beschriebenen Anweisungen exakt ausführt. Formalismen der Wissenschaften sind damit eine Form von Handlungstheorie. Holling und Kempin bezeichnen sie als 'implementierte Theorie'. Unter einer 'Implementierung' versteht man dabei die Umsetzung eines formalen Verfahrens – also eines Algorithmus' – in eine Maschine. Der Begriff Implementierung, der in den Computerwissenschaften sehr gängig ist, kommt aus dem Englischen und bedeutet soviel wie, etwas ausführen oder in Kraft setzen. In der Informatik ist dies der Schritt von einem Algorithmus zu dessen Umsetzung als Programm in einer bestimmten Programmiersprache. Während der Algorithmus – in diesem Bild – die Theorie eines Prozesses darstellt, ist die Implementierung die Verwirklichung dieser Theorie als Maschine. Damit eine Theorie implementierbar ist, muss sie operationalisierbar sein, d.h. sie muss in Form von Handlungsbegriffen beschrieben

33 In: Geistige und körperliche Arbeit, S. 21f.

sein. Damit wären implementierbare Theorien eine Teilmenge aller möglichen Theorien, die sich dadurch auszeichnen, dass sie operationalisierbar sind. Holling und Kempin stellen allerdings die These auf, dass die implementierbare Theorie der herrschende Theorietypus des abendländischen Denkens ist und sich bereits in den aristotelischen Syllogismen ausdrückt.[34] Die gesamte abendländische Logik als implementierbare Theorie aufzufassen ist allerdings erst durch die Verfügbarkeit der programmierbaren Maschine möglich. Durch die Entwicklung formaler Modelle und Kalküle war ein wesentlicher Schritt bereits geleistet. Formale Modelle erlauben es, inhaltliche Probleme und Aufgaben als operationalisierte Verfahren zu beschreiben, die damit prinzipiell auch von einer Maschine ausgeführt werden können. Die Sprache, Probleme zu beschreiben, liefert die abendländische Logik, die durch Boole ihre vollständige Formalisierung als Handlungsprozess erfahren hat. Was dann noch fehlt, ist eine Maschine, die in der Lage ist, sämtliche formalen Kalküle automatisch ablaufen zu lassen. Durch Alan Turing wurde diese Maschine 1936 erstmals beschrieben. Es dauerte dann allerdings noch einige Jahre, bis diese Maschine nicht nur als Papiermaschine vorlag, sondern auch als gebaute und funktionsfähige Maschine zur Verfügung stand.[35] Die These von Holling und Kempin wird nicht zuletzt dadurch gestützt, dass heute Naturwissenschaften vollständig auf der Basis der programmierbaren Maschine betrieben werden. Die Mächtigkeit dieses Prinzips der implementierten Theorie wird häufig unterschätzt. Eine oft vorgetragene Kritik ist verbunden mit der Vorstellung, dass man auf der Basis operationalisierter Formalismen nicht mehr herausbekommen kann, als man bereits hineingesteckt hat. So ist etwa bei Adorno zu lesen:[36] „Aber Denken ist gleichzeitig mit seiner Verselbständigung zur Apparatur Beute von Verdinglichung geworden, zur selbstherrlichen Methode geronnen. Grob offenbart sich das an den kybernetischen Maschinen. Sie stellen den Menschen die Nichtigkeit des formalisierten, seiner Sachgehalte entäußerten Denkens vor Augen, insofern sie manches von dem, woran die Methode subjektiver Vernunft ihren Stolz hatte, besser vermögen als die denkenden Subjekte. Machen sich diese leidenschaftlich zu Vollzugsorganen solcher Formalisierung, so hören sie virtuell auf, Subjekt zu sein. Sie nähern sich den Maschinen als ihr unvollkommeneres Abbild. ... Philosophisches Denken beginnt erst, sobald es sich nicht damit begnügt mit Erkenntnissen, die sich absehen lassen und bei denen nicht mehr herausschaut, als man schon hineinsteckte. Der menschenwürdige Sinn der Computer wäre es, das Denken der Lebendigen so sehr zu entlasten, daß es Freiheit gewinnt zu dem nicht schon impliziten Wissen.“ Hier wird ein sehr wichtiger Punkt im Zusammenhang mit formalen Beschreibungen angesprochen. In der Tat ist durch den Formalismus bereits der vollständige Ablauf determiniert, er spannt den Rahmen auf, innerhalb dessen sich die Bearbeitung von Eingabegrößen bewegt. Nach Ausführung eines

34 Vgl. Holling, Kempin, S. 82.

35 Siehe auch Zwischenspiel: Universelle Turing Maschinen & minimale Kontrollstrukturen.

36 In: Anmerkungen zum philosophischen Denken, in: Stichworte. Kritische Modelle 2, S. 11.

Schrittes steht eindeutig fest, welcher Schritt als nächstes ausgeführt wird. In diesem Sinne ist auch das Diktum Heinz von Foersters zu verstehen „Nur die Fragen, die im Prinzip unentscheidbar sind, können wir entscheiden.“[37] Die entscheidbaren Fragen sind die formalisierbaren und programmierbaren Fragen. Aber durch die Wahl des formalen Rahmens sind die Fragen bereits entschieden, egal, ob wir die einzelnen Folgen, die sich aus der Wahl des Rahmens ergeben, schon sehen können oder ob wir sie erst langwierig von Hand oder maschinell ableiten müssen. Nach einer Folge stringenter logischer Schlüsse erzielen wir ein definitives ‘Ja’ oder ‘Nein’.[38] Dem Denken in derartigen Kausalketten fehlt vollständig die Dimension des Werdens, da die Zukunft jeweils vollständig in der Gegenwart enthalten ist. Als Beispiel für eine unentscheidbare Frage nennt von Foerster die Frage nach dem Ursprung des Universums, die in jeder Kultur anders entschieden wird. Wir stehen hierbei nicht unter irgendeinem Zwang, auch nicht dem Zwang der Logik, wenn wir prinzipiell unentscheidbare Fragen entscheiden. Hierauf zielt auch Adornos Aussage über den eigentlichen Wert philosophischen Denkens ab. Dennoch ist das Problem nicht so einfach zu fassen, wie Adorno es sieht. Wir werden weiter unten sehen, dass durchaus „mehr herausschauen kann, als man hineingesteckt hat“, insbesondere interaktive sowie evolutionäre und lernende Algorithmen sind mit diesem einfachen Bild nicht mehr zu fassen. Der Rahmen verschiebt sich hier während des Ablaufs und durch den Ablauf selbst.

1.2.5 Programmierbarkeit im engeren Sinn

„Der wesentliche Maschinenbestandteil des Computers ist sein Programm; Programmieren bedeutet, Maschinen zu konstruieren.“

Arno Bammé et. al.[39]

Wir haben im Vorangegangenen versucht deutlich zu machen, dass neben den Fortschritten in der handwerklichen Mechanik vor allem die rationalistische Tradition und die damit einhergehende Formalisierung wissenschaftlicher Ergebnisse und technischer Abläufe wesentliche Voraussetzungen für die heutige programmierbare Maschine bilden. In diesem Abschnitt wollen wir uns mit der programmierbaren Maschine im engeren Sinn – dem Computer – beschäftigen, d.h. mit Maschinen, deren Funktionalität durch Software festgelegt wird. Der Begriff Computer setzt tatsächlich Programmierbarkeit voraus. Wir sprechen deshalb bei den Erfindungen von Schickard, Pascal und Leibniz von Rechenmaschinen und nicht von Computern, da hier die arithmetischen Funktionen durch die

37 Vgl. KybernEthik, S. 73.

38 Die Klasse der partiell entscheidbaren Probleme bildet eine Zwischenform zwischen entscheidbaren und unentscheidbaren Fragen. Im positiven Fall kommt der Algorithmus zu einer Entscheidung und wird anhalten. Im negativen Fall wird die Berechnung nicht stoppen, d.h. die Frage wird nicht entschieden.

39 In: Technologische Zivilisation und die Transformation des Wissens, S. 42.

Maschinen starr vorgegeben und mit der Apparatur verschmolzen waren. Wir können den Weg hin zur Programmierung, wie wir sie heute kennen, allerdings nur skizzieren. Dieser grobe Durchlauf durch verschiedene historische Stufen der Programmierung soll es in der Folge erlauben, die Mächtigkeit der universellen Programmierbarkeit besser zu verstehen.

Der moderne Maschinenbegriff zeichnet sich dadurch aus, dass die materielle Realisierung der Maschine vollkommen offen gehalten wird, sie kann mechanisch, elektronisch, optisch oder sonst wie ausgeführt werden. Insbesondere müssen Maschinen nicht mehr entsprechend der ursprünglichen Bedeutung des Begriffs aus sichtbaren beweglichen Einzelteilen zusammengesetzt sein. Die kybernetische Maschine abstrahiert vollkommen von der materiellen Form ihrer Realisierung. Zwei Maschinen gelten dann als identisch, wenn sie dasselbe Verhalten generieren. Im Falle des Computers heißt das, wenn sie dieselbe Funktion berechnen. In diesem Zusammenhang weist bereits Karl Marx darauf hin, dass die Manufaktur eine aus 'Menschenmaterial' zusammengesetzte Maschine ist. Auch Lewis Mumford argumentiert ähnlich, wenn er behauptet, die Pyramiden wurden durch eine antike Megamaschine gebaut. Ob eine Maschine aus Arbeitern, aus Rädchen und Hebeln, aus Stahl oder aus elektronischen Schaltkreisen zusammen gesetzt ist, spielt für unseren modernen Maschinenbegriff keine Rolle. Wichtig ist ausschließlich die Funktion, die von der Maschine ausgeführt wird. Maschinen sind Gebilde aus abstrakten Elementen, die so kontrolliert werden und ineinander greifen, dass ein gewünschtes Verhalten entsteht.

1.2.5.1 Analoge Programme

Von 'Programmierbarkeit' können wir im Zusammenhang mit unserem abstrahierten Maschinenbegriff sprechen, wenn wir das konkrete Verhalten der Maschine ändern können, ohne nochmals in die innere Struktur der Maschine einzugreifen, d.h. ohne nochmals ihren Konstruktionsplan zu ändern. Der Entwurf der Maschine muss das Spektrum der Verhaltensmöglichkeiten der Maschine bereits vorgesehen haben, das konkrete Verhalten darf aber jeweils nur eine Ausblendung aus diesen Möglichkeiten sein. Das gewünschte Verhalten kann der Maschine eingeschrieben werden, ohne ihr Konstruktionsprinzip zu verändern. Die älteste Form mechanischer Programmierbarkeit, die wir bisher kennen gelernt hatten, war der fahrende Tempelautomat des Heron von Alexandria. Durch Variation der Schnurwicklung konnten unterschiedliche Verhaltensweisen des Automaten 'programmiert' werden. Der Entwurf der Maschine sieht diese unterschiedlichen Verwendungsformen der Maschine bereits vor. Eine weitere Urform der Programmierbarkeit waren die Androiden der Familie Jaquet-Droz. Hier konnten durch Auswechselung von Sätzen konischer Nocken ebenfalls Verhaltensmodifikationen durchgeführt werden. Der Rahmen der unterschiedlichen Verhaltensweisen ist im Falle des fahrenden Tempelautomaten natürlich noch sehr eingeschränkt. Die Automaten der Familie Jaquet-Droz bieten

hier bereits – zumindest prinzipiell – weitaus mehr Möglichkeiten. Auch wenn zum Beispiel beim Zeichenandroiden tatsächlich nur vier Zeichnungen realisiert waren, könnte im Prinzip durch Erstellen weiterer 'Programme' (Nockensätze) das Verhaltensspektrum erweitert werden. In beiden Fällen – dem Tempelautomaten wie den Androiden – handelt es sich um analoge Steuerungen. Das Programm steuert in einem stetigen Prozess die Bewegungen der Maschinen. Aber auch in der Rechentechnik gab es zu Beginn der Computerentwicklung in den 1940-er Jahren eine Vielfalt an analogen programmierbaren Maschinen. Bis weit in die 1960-er Jahre gehörten Analogcomputer zur Standardausstattung vieler Forschungs- und Ingenieurgruppen und es gab eine ganze Reihe von Herstellern für diese Maschinen. Eine berühmte Maschine war der *Differential Analyzer*, ein mechanischer Analogcomputer, den Vannevar Bush am MIT zur Lösung von Differentialgleichungssystemen entwickelt hatte. Der *Differential Analyzer* musste von Hand programmiert werden, mit einem Schraubschlüssel in der einen Hand und einem Schalthebel in der anderen.[40] Wir wollen diesen interessanten Zweig der Programmierung an dieser Stelle nicht weiter verfolgen, da der hier im Zentrum stehende Algorithmenbegriff unter anderem durch diskretes, d.h. schrittweises Vorgehen gekennzeichnet ist.[41] Während die frühen analogen Automaten und Maschinen unendlich viele Zwischenstellungen einnehmen können und durch unkontrollierbare Schwankungen charakterisiert sind, setzen Implementierungen von diskreten Algorithmen Maschinen voraus, die nur eine endliche Anzahl von Stellungen einnehmen können. Diese möglichen Stellungen heißen im Allgemeinen 'Zustände' der Maschine.

1.2.5.2 Programme als Code

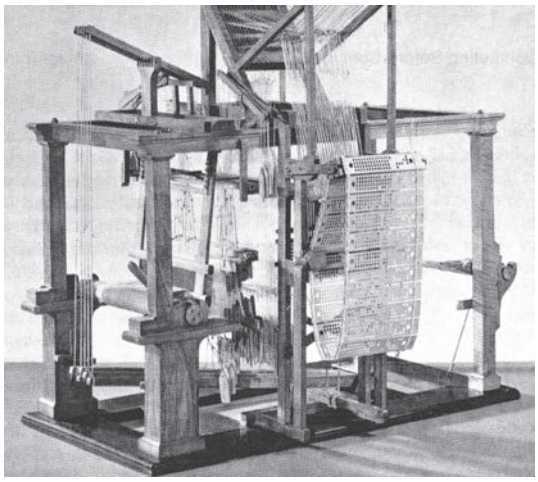
„Das Geheul der Leierkästen, das Mr. Babbage, Fellow der Royal Society, die Ohren durchbohrt, ist programmgesteuert. Ein Satz Lochkarten dudelt, bestimmt den Stückakkord, und hält den Krankenstand fest."
aus Charles Babbage 1792 – 1871,
von Hans Magnus Enzensberger[42]

Die Idee code-gesteuerter Maschinen, die algorithmisches Verhalten realisieren, finden wir erstmals in der lochkartenkontrollierten *Analytischen Maschine* von Charles Babbage. Erst durch den Code wird eine Ablösung des Programms von der Hardware eingeleitet. Schon mehr als 30 Jahre vor Babbage hatte J.M. Jacquard bei seinen um 1800 erfundenen Webstühlen Lochstreifen eingesetzt, die das Weben unterschiedlicher Stoffmuster automatisch steuerten. Aber sogar vor der Verwendung in Webstühlen wurden einzelne Musikinstrumente

40 Vgl. Arthur W. Burks, From the ENIAC to the Stored-Program Computer: Two Revolutions in Computers, in: N. Metropolis, et. al. (Hrsg.), A History of Computing in the Twentieth Century, 1980.

41 Siehe Zwischenspiel: Algorithmisches Denken.

42 In: Elixiere der Wissenschaft, S. 38.



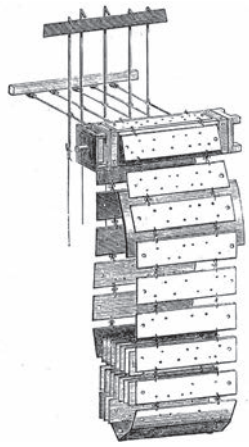


Abbildung 18/19:
Jacquard-Webstuhl mit Lochstreifensteuerung (links). Die 'Software' des Jacquard-Webstuhls. Gelochte Pappkarten steuern das Weben der Muster (rechts).

gebaut, die kontrolliert durch Lochstreifen verschiedene Musikstücke aufführen konnten.

Lochkarten sollten zur tragenden Idee der Automatisierung im 19. Jahrhundert werden. Die Lochkarte fungierte quasi als erster maschinenlesbarer Datenträger. Zu diesem Zweck griff der Amerikaner Herman Hollerith die Idee auf und konstruierte die erste Lochkarten-Rechenmaschine anlässlich der statistischen Auswertung der Volkszählung in den USA 1890. Aufgrund des großen Erfolges wurden im Anschluss auch in Österreich und Kanada und im Jahre 1897 auch in Russland Volkszählungen mit Hollerith-Maschinen durchgeführt. Die *Analytical Engine* von Charles Babbage ist allerdings viel interessanter als die Hollerithsche Maschine, bei der die Lochkarte nur als Datenträger verwendet wurde.[43] Die mit Babbage befreundete Ada Augusta Countess of Lovelace sagte über Babbages Analytische Maschine: „Die Analytical Engine webt algebraische Muster, so wie Jacquards Webstuhl Blumen und Blätter webt." Wenn wir unter einem Programmcode die Abbildung einer endlichen Menge von Zeichen in Signalfolgen der Maschinen verstehen, durch die Zustandsübergänge der Maschine gesteuert werden, dann wird klar, dass Babbages Maschine unseren heutigen Begriff von Programmierbarkeit bereits sehr nahe kommt. Bestimmte Lochkombinationen repräsentieren hier ganz bestimmte Operationen. Die einzelnen Karten konnten mit Bändern zu einem Programm zusammengebunden werden. Der Lochkartenleser würde jeweils eine Karte lesen und spezielle 'microprogram barrels' würden die Maschine dann veranlassen, die entsprechende Operation auszuführen. Sogar einfache Schleifen, d.h. variierte Wiederholungen eines

43 Siehe auch Zwischenspiel: Universelle Turingmaschinen & minimale Kontrollstrukturen.

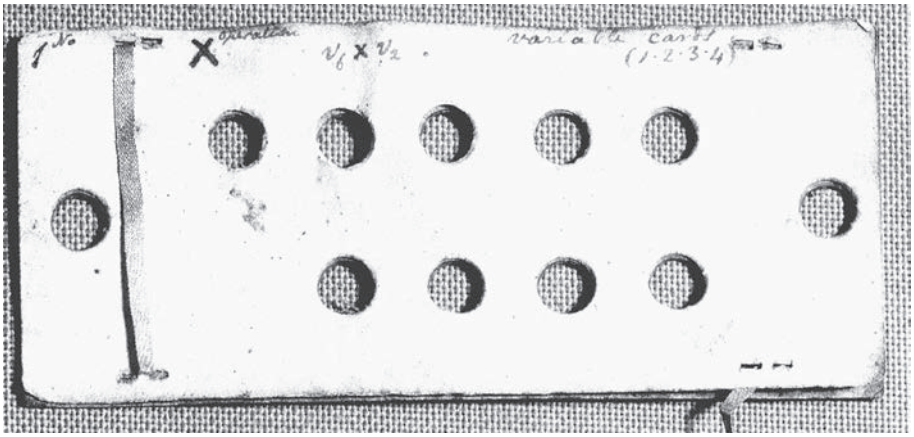


Abbildung 20:
Beispiel einer Babbage-Lochkarte. Die Karte enthält die Instruktionen für die Multiplikation des Inhalts der Speicherzelle V6 mit dem Inhalt der Speicherzelle V2.[44]

Bearbeitungsschemas, waren auf diese Weise bereits realisierbar. Eine Karte konnte Anweisungen enthalten, wie: Wenn die Zählapparatur eine Zahl geringer als N enthält, dann gehe P Karten zurück und starte den Vorgang erneut.[45]

1.2.5.3 Steckfeld-Programmierung

Gewissermaßen eine Zwischenform zwischen stetigem Verhalten und diskreter Steuerung sind Durchschaltprinzipien in Kommunikationssystemen, wie sie zu Beginn des 20. Jahrhunderts in Fernsprechämtern oder auch in heute noch gebräuchlichen Kreuzschienenverteilern zu finden sind, die als Steckfelder realisiert werden. Mittels einfacher Steckverbindungen können hier verschiedene Ein-/Ausgangs-Verknüpfungen eines Tableaus variiert werden. Durch die Verbindung von Kreuzungspunkten des zweidimensionalen Rasters lässt sich eine endliche Anzahl von Zuständen des Verteilers realisieren. Das eigentliche Verhalten des Systems, d.h. die Nachrichten, die durch die Input/Output-Verbindungen laufen, können wieder stetige Signale sein. Entscheidend ist, dass auch hier die möglichen Funktionen bereits beim Entwurf und beim Bau der Maschine geplant wurden. Wir wollen bei dieser Variante allerdings mehr von Konfigurationen einer Vorrichtung als von Programmierungen sprechen, da das wählbare Verhalten durch entsprechende mechanische Einstellungen realisiert wird und nicht durch Codes.

In den frühen IBM Computern der 1940-er Jahre wurde die Idee der Lochkarte mit der Idee der Steckfeldprogrammierung verbunden.[46] Die Ein- und Ausgabe wurde mit Hilfe von Lochkarten realisiert und das Programm wurde über

44 Vgl. Michael R. Williams, History of Computing Technology, S. 184.

45 Ebd.

46 Quelle: Columbia University Computing History, http://www.columbia.edu/acis/history/.

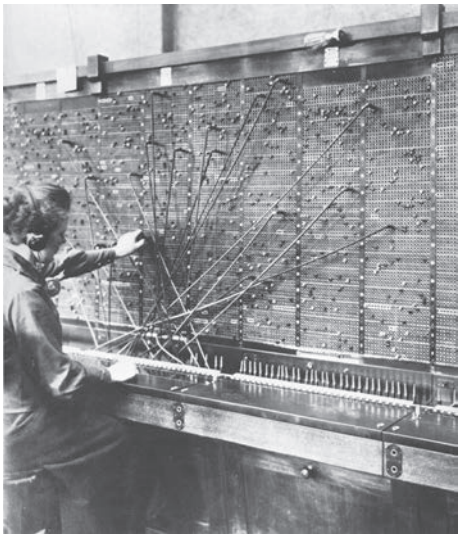


Abbildung 21:
Arbeitsplatz der 'Klingelfee'.

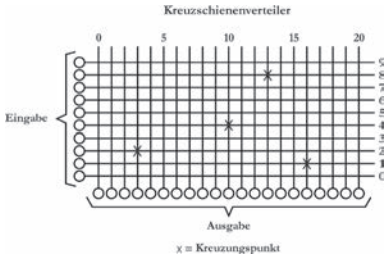


Abbildung 22:
Schema eines Kreuzschienenverteilers. Durch wahlweise Verbindung von Kreuzungspunkten (x) kann eine Vielzahl neuer Ein- / Ausgabekonfigurationen realisiert werden.

Steckfelder eingegeben. Zum Beispiel konnte die elektromechanische *IBM Type 602* (*calculating punch*) aus dem Jahre 1946 die vier Grundrechenarten (Addition, Subtraktion, Multiplikation, Division) mit einer Verarbeitungsgeschwindigkeit von einhundert Karten pro Minute durchführen. Die Daten wurden über einen Lochkartenleser eingelesen und die Ergebnisse der Verarbeitung entweder direkt auf die Eingabekarten oder auf nachfolgende Karten gestanzt. Die Verarbeitung der Daten wurde durch das Steckfeld-Programm (*plugboard program*) kontrolliert. Die meisten IBM Lochkartengeräte – Buchhaltungsmaschinen, Tabulatoren, Rechenmaschinen – wurden über derartige Kontrollfelder gesteuert. Die einzelnen Instruktionen wurden realisiert, indem die Steckmuffen (*hubs*) mit

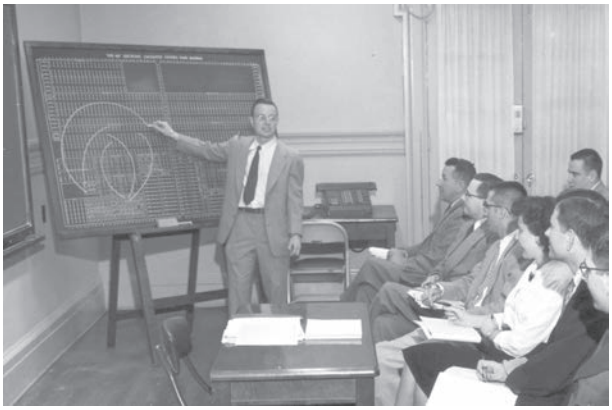


Abbildung 23:
Programmierkurs um ca. 1950. Eric Hankam unterrichtet einen dreiwöchigen Computerkurs am Watson Laboratory der Columbia University in New York City. Die Programme konnten direkt mit Kreide auf Tafeln mit aufgedrucktem Steckfeld gezeichnet werden.

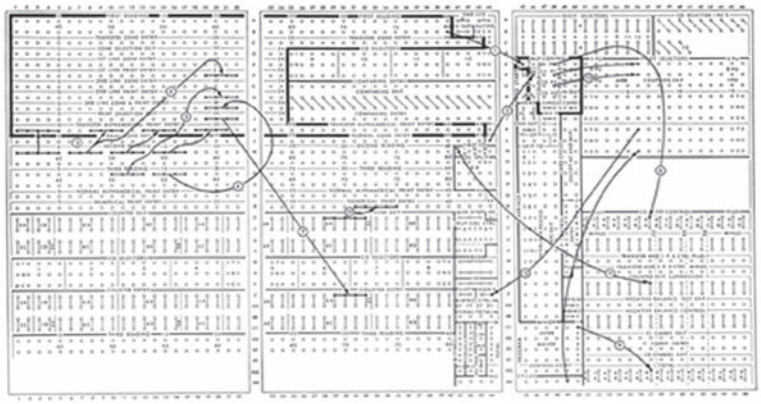


Abbildung 24:
IBM 402/403 Steckfeld-Diagramm (*Plugboard*) mit eingezeichneten Verbindungen.

Kabeln verbunden wurden. Der Benutzer konnte damit unter anderem genau spezifizieren, welche Spalten der Lochkarte gelesen oder gestanzt und welche Felder welchen Akkumulatoren zugeführt wurden. Eine einfache Aufgabe konnte etwa darin bestehen, Zahlenspalten auszudrucken, die von einem Stapel Karten gelesen wurden. In diesem Fall würden eine Reihe von Kabeln die entsprechenden Kartenspalten mit den entsprechenden Druckerspalten verbinden.

1.2.5.4 Maschinen mit Programmspeicher

Der letzte Schritt zum programmierbaren Universalrechner, wie er auch im Zentrum unserer Betrachtungen in diesem Buch steht, ist die Programmspeicherung, d.h. die Steuerung der Maschine durch ein Programm, das zusammen mit den zu verarbeitenden Daten im internen Speicher der Maschine gehalten wird. Zwar gibt es Hinweise darauf, dass bereits Babbage an eine solche Steuerung gedacht haben könnte, gleichzeitig ist es vom heutigen Standpunkt aus anhand seiner wenigen Aufzeichnungen schwierig zu beurteilen, ob Babbage überhaupt eine klare Vorstellung vom Begriff des Programms hatte. Die Idee des Programmspeichers findet sich unzweifelhaft erstmals bei Konrad Zuse, wurde von ihm aber offensichtlich nicht weiter verfolgt.[47] Zuse ist aber vor allem auch deshalb ein wichtiger Pionier der Programmierung, weil sein *Plankalkül* bereits eine umfassende Programmiersprache darstellt, die versucht beliebige Rechenvorschriften rein formal zu beschreiben. Theoretisch gefasst wird das Prinzip der Gleichbehandlung von Daten und Programm erstmals bei Alan Turing in seinem Paper „On Computable Numbers, with an Application to the Entscheidungsproblem“. Sowohl die Daten als auch das Programm stehen bei Turing auf dem Band der Maschine und sind damit einer Manipulation prinzipiell gleichwertig zugänglich. Trotz der parallelen Existenz all dieser Überlegungen, die

47 Vgl. Brian Randell, The Origins of Digital Computers, S. 375 f.

Abbildung 25:
Konrad Zuse an der Programmleseeinheit des Z4. Die Instruktionen für die Maschine wurden auf ausrangierten 35mm Filmstreifen codiert.

vollkommen unabhängig von unterschiedlichen Personen zu unterschiedlichen Zeiten angestellt wurden, scheint es vernünftig, den durch John von Neumann 1945 im „First Draft of a Report on the EDVAC" beschriebenen Entwurf als die erste speicherprogrammierte Maschine zu bezeichnen. Die Idee der Programmspeicherung mag oberflächlich betrachtet relativ bescheiden erscheinen, unsere weiter unten folgenden Überlegungen werden jedoch zeigen, dass die Gleichbehandlung von Daten und Programm in einem gemeinsamen Speicher folgenreiche theoretische wie praktische Bedeutungen besitzt.

1.2.5.5 Universelle Kontrollstrukturen

Damit war die universell programmierbare Maschine zwar – technisch gesehen – erfunden, aber erst mit der Entwicklung problemorientierter Programmiersprachen in den 50-er Jahren beginnt die Ablösung der Programme von den Bedingungen der Hardware und damit Softwareentwicklung als wirklich eigenständige Disziplin[48]. Das Vermittlungsprogramm, das ein hochsprachlich verfasstes Programm als ganzes in Maschinensprache übersetzt, ist der 'Compiler'. Der 'Interpreter' dagegen übersetzt Befehlszeile für Befehlszeile und führt jede

48 Siehe auch Zwischenspiel: Programmierparadigmen und die Entstehung von Java.

Anweisung sofort aus. Eine Programmiersprache heißt 'problemorientiert', wenn sie erlaubt, Algorithmen unabhängig von einer bestimmten Rechenanlage zu beschreiben. Viel wichtiger ist aber die zweite Eigenschaft problemorientierter Sprachen, die – wie der Name schon sagt – darin besteht, dass sie sich in der Schreib- und Sprechweise an die Anwendungsbereiche der Programmiersprache anlehnen. Im Bereich technisch-wissenschaftlicher Anwendungen gehörten FORTRAN und ALGOL zu den ersten problemorientierten Sprachen, im kommerziell-administrativen Bereich war dies COBOL. Entsprechend der Anwendungsanforderungen legte zum Beispiel COBOL mehr Gewicht auf die Strukturierung der Daten, ALGOL dagegen auf die Strukturierung der Algorithmen. Es sind tatsächlich nur drei Kontrollmechanismen erforderlich, um sequentielle Abläufe universell zu beschreiben. Sie bilden – in der jeweils einen oder anderen Variante – den Kern höherer Programmiersprachen und damit auch der gesamten Algorithmentheorie. Diese drei Prinzipien sind 1. die Sequenz, 2. die Schleife und 3. die Fallunterscheidung.

1. Die Sequenz
   Die einfachste Form der Gliederung eines Arbeitsprozesses ist seine Zerlegung in einzelne elementare Arbeitsschritte. Der Gesamtprozess besteht dann aus der Aneinanderreihung – einer Sequenz – solcher Elementareinheiten. Jede Tätigkeit, ob im Handwerk, der industriellen Fertigung oder im Verwaltungsprozess, zerfällt ganz natürlich in unterscheidbare und benennbare Einzelaktivitäten. In der industriellen Fertigung wurde dieses Prinzip gezielt genutzt, um effiziente Produktionsstrukturen aufzubauen. Die einfachste Form der Prozesssteuerung in der Programmierung ist folglich die Hintereinanderschreibung gewisser Elementarschritte.
2. Die Schleife (loop)
   Weiteres wesentliches Kennzeichen von Arbeitsprozessen ist die Wiederholung. Eine bestimmte Folge von Arbeitsschritten muss immer wieder durchlaufen werden. Die einzelnen Arbeitsschritte sind hierbei jeweils gleich, dennoch ist jeder Durchlauf im Hinblick auf das bearbeitete Material verschieden. Mit jedem Schritt erfolgt in der Regel eine sukzessive Veränderung des Materials. Eine Ausnahme bilden hier Kontrollroutinen, die solange vollkommen identisch und ohne Ergebnis ablaufen, bis sich die Bedingungen durch externe Einflüsse ändern. Erst dann werden andere Routinen aktiviert, die auf diese Änderungen hin abgestimmt sind. Ein Äquivalent hierzu im Bereich der Programmierung sind zum Beispiel die Elementarschleifen interaktiver Anwendungen. Hier wird zyklisch abgefragt, ob eine Eingabe durch den Benutzer stattgefunden hat und erst, wenn dies tatsächlich der Fall ist, werden die entsprechenden Bearbeitungsroutinen gestartet.
3. Die Beurteilung (Fallunterscheidung, if-Konstrukt)
   Das letzte und wichtigste Kontrollelement zur Steuerung sequentieller Abläufe ist die Fallunterscheidung. Während des Arbeitsvorgangs muss immer wieder

geprüft, und aufgrund dieser Prüfung entschieden werden, wie weiter verfahren werden soll. Es reicht in der Regel nicht, wenn wir eine Wiederholstruktur verwenden, die unabhängig vom Verlauf eines Arbeitsvorgangs eine feste Anzahl von Wiederholungen vorsieht. Diese Vorgehensweise ist auch im Hinblick auf die Programmierung wesentlich. 'Universalität' ist tatsächlich erst gewährleistet, wenn bei Eintritt in eine Wiederholschleife nicht schon bekannt ist, wie oft diese durchschritten wird. Diese Wiederholzahl wird erst in der Berechnung anhand von Zwischenergebnissen bestimmt.[49] Die Fallunterscheidung, als gleichfalls elementare Einheit des Denkens, ist das zentrale Element, um Prozessabläufe zu variieren.

In der Berechenbarkeitstheorie konnte gezeigt werden, dass es verschiedene programmiersprachliche Umsetzungen dieser drei Elemente gibt, die alle Universalität garantieren. Die minimalen Anforderungen an eine universelle Programmiersprache sind, bezogen auf die Kontroll- und Datenstrukturen, relativ gering. Neben den drei Kontrollmechanismen, müssen wir als minimale Datenstruktur die 'natürlichen Zahlen' zur Verfügung stellen und als Operationen die Addition und Subtraktion.[50] Während die früher übliche Goto-Anweisung inzwischen als Sprachkonstrukt verschwunden ist, bieten moderne Sprachen verschiedene – gewissermaßen redundante – Möglichkeiten zur Beschreibung des Programmablaufs an. Wir sehen, dass auf dieser Ebene der Programmbeschreibung die formale Logik, die wir im Vorangegangenen ausführlich besprochen haben, in den Hintergrund getreten ist. Hardwareseitig werden die Operatoren und Kontrollstrukturen der Programmiersprachen aber mit Hilfe logischer Schaltungen realisiert. Die meisten problemorientierten Sprachen bieten zudem auch in ihrem Sprachvorrat logische Operatoren für die Verknüpfung von Wahrheitswerten an. Wichtig ist, dass mit der permanenten Weiterentwicklung der Programmiersprachen der Eigenwert und die Eigengesetzlichkeit von Programmstrukturen erkannt wurde. Mit den formalen Kontrollstrukturen und dem zugehörigen Begriff des Prozesses als Aktivitätsträger von Abläufen, haben zeitliche Arbeitsvorgänge ihren eigenen *Logos* erhalten. Erst Logik plus Kontrolle ergibt Algorithmen.

49 Siehe Zwischenspiel: Universelle Turingmaschinen & minimale Kontrollstrukturen.

50 Ebd.

1.2.5.6 Die universell-programmierbare Maschine

„Es war Kants Verdienst zu erkennen, daß dieser Zwang [zu mechanistischen, unpersönlichen Erklärungen] in uns und nicht in den Dingen ist. [Und] es war Weber, der erkannte, dass es der Geist auf einer bestimmten historischen Stufe und nicht der Geist als solcher ist, der diesem Zwang unterliegt."
Ernest Gellner[51]

An diesem Punkt sind wir in der Lage, wesentliche Charakteristika der 'Programmierbarkeit im engeren Sinne' zusammenzufassen. Die Beschränkung auf die wenigen vorangegangenen historischen Anmerkungen reicht aus, um die grundlegenden Eigenschaften der heutigen programmierbaren Maschine abschließend deutlich zu machen:

- *Maschinen als formale Zeichenprozesse*
  Wir hatten schon weiter oben festgestellt, dass der moderne Maschinenbegriff von der Realisierung der Maschine – also dem verwendeten Material und der konkreten Konstruktion – abstrahiert. Entscheidend sind nur noch die Ein-/Ausgaberelationen und die internen Zustandsübergänge der Maschine. Die Wirkungsweise der Maschine kann damit vollständig als Funktion (bzw. als formale Prozedur) beschrieben werden. Im Hinblick auf die Grundprinzipien algorithmischer Maschinen ist bei der Ablösung des mechanischen Zeitalters durch das elektronische nichts passiert. Die gleichen algorithmischen Denkmuster manifestieren sich lediglich in einem neuen Material. Es braucht allerdings einen Aktivitätsträger, der die Handlungsanweisungen tatsächlich ausführt. Ob dieser Aktivitätsträger eine elektronische Vorrichtung, eine mechanische oder eine organische ist, bleibt zweitrangig. Turing selbst formuliert diesen Sachverhalt folgendermaßen: „Es ist möglich, den Effekt einer Rechenmaschine zu erreichen, indem man eine Liste von Handlungsanweisungen niederschreibt und einen Menschen bittet, sie auszuführen. Eine derartige Kombination eines Menschen mit geschriebenen Instruktionen wird 'Papiermaschine' genannt. Ein Mensch, ausgestattet mit Papier, Bleistift und Radiergummi sowie strikter Disziplin unterworfen, ist in der Tat eine Universalmaschine."[52] Bezüglich der Art und Weise, wie diese Handlungsanweisungen niedergeschrieben sein müssen, gibt es gewisse Freiheiten. Die Theorie der Berechenbarkeit hat eine ganze Reihe von Formalismen entwickelt, die sich schließlich als gleichmächtig erwiesen haben und die alle in der Lage sind, den Begriff des Algorithmus zu präzisieren.[53] Jede höhere Programmiersprache ist ebenfalls ein solcher allgemeiner Formalismus. Der Begriff der Maschine und der Begriff des Algorithmus erweisen sich aus

51 Zitiert nach: Richard Tarnas, Idee und Leidenschaft: Die Wege des westlichen Denkens, S. 529.

52 In: Intelligent Machinery, zit. nach: Alan M. Turing, Intelligence Service, S. 91.

53 Siehe Zwischenspiel: Algorithmisches Denken.

Sicht eines Programmierers als Äquivalent. Sollen zwei Maschinen hinsichtlich ihrer Struktur verglichen werden, so versucht man dies durch Angabe einer Abbildung – also wiederum einer Funktion –, durch welche die eine Maschine in die andere überführt wird. Ist diese Abbildung nur in die eine Richtung möglich, so spricht man von Homomorphie, ist sie umkehrbar-eindeutig, d.h. in beiden Richtungen gültig, so heißt sie Isomorphie. Der Kybernetiker Ross Ashby beschreibt dies folgendermaßen:[54] „Die kanonischen Darstellungen von zwei Maschinen (oder Black-Boxes, d.h. von zwei Sachen, die sich genau so verhalten wie eine geschlossene eindeutige Transformation) sind isomorph, wenn eine umkehrbar-eindeutige Transformation von den Zuständen (Eingang und Ausgang) der einen Maschine zu denen der anderen Maschine die eine Darstellung in die andere umwandeln kann." Das heißt, zwei Maschinen gelten als strukturgleich, wenn sie sich zwar in der Art der Elemente (Zustände, Alphabete, etc.) unterscheiden, nicht aber im Verhältnis der Elemente untereinander und ihrem Ein-/Ausgabe-Verhalten. Nachdem die Realisierung von Maschinen als formale Zeichenprozesse erreicht war, stellte sich natürlich auch bald die Frage nach der Effizienz ihrer Beschreibung. Die Entwicklung höherer Programmiersprachen erlaubt es, Maschinen zu beschreiben, die näher am Problem und am Denken des Programmierers liegen. Befehle werden nicht mehr direkt als Löcher in Karten codiert, sondern auf der Basis von Zeichenalphabeten sprachlich beschrieben. Ein Vermittlungsprogramm, der Compiler, übersetzt die Zeichen eines Programms dann in maschinenlesbare Befehle. Die Realisierung von Compilern markiert damit eine weitere wichtige Stufe, die es erlaubt, Maschinen auf unterschiedliche Weisen sprachlich zu fassen und sie auf derselben Hardware ablaufen zu lassen.[55]

- *Die Zweiteilung der Maschine*
  Die programmierbare Maschine setzt sich damit immer aus zwei Maschinenkonzepten zusammen: 1. Der universellen Hardware, die noch offen ist für die Einschreibung eines konkreten Verhaltens und die prinzipiell in der Lage ist, jede andere Maschine zu simulieren. 2. Der universellen Programmiersprache, d.h. einer Sprache, in der konkrete Maschinen beschrieben werden können. Die einzelnen in dieser Sprache beschriebenen Programme werden in den Speicher der Maschine gelesen und anschließend ausgeführt. Die Hardware wird sich dann exakt so wie die durch die Software beschriebene Maschine verhalten. Die erste Maschine – die Hardware – ist in der physikalischen Welt verhaftet. Sie verbraucht Energie, muss physikalischen Gesetzen gehorchen, unterliegt Alterungsprozessen und kann Defekte zeigen. Die zweite Maschine – das Programm – unterliegt dagegen nicht mehr den materiellen Gesetzen der Realwelt. Die einzige Begrenzung ist unser Denken

54 In: Einführung in die Kybernetik, S. 148.

55 Siehe Zwischenspiel: Programmierparadigmen und die Entstehung von Java.

und unsere Vorstellungskraft sowie die Begrenzungen der Programmiersprache. Wir können aufsetzend auf der ersten Maschine neue Maschinen simulieren und untersuchen, die physikalisch nicht mehr zu bauen wären. Das ist der entscheidende Hebel, der es uns ermöglicht, den Computer als Werkzeug der Imagination zu nutzen. Dies bedeutet eine wesentliche Erweiterung des Gedankens, den Turing noch mit seiner Universalmaschine verband. Er sah den Nutzen der programmierbaren Universalmaschine darin, das Problem, unterschiedliche Maschinen für unterschiedliche Aufgaben zu konstruieren, zu ersetzen durch die Schreibarbeit, die erforderlich ist, diese Maschine für eine bestimmte Aufgabe zu programmieren. Universalität ist in diesem Zusammenhang gegeben, wenn die Maschine in der Lage ist, jede andere denkbare und ausreichend präzise beschreibbare Maschine zu simulieren. Beide Konstrukte – die Hardware wie die Programmiersprache – müssen bestimmte Mechanismen in ihrem inneren Aufbau vorsehen, damit Universalität gewährleistet ist. Es gibt ein minimales Niveau an Kompliziertheit, unterhalb dessen die Eigenschaft der Universalität verloren geht.[56] Ist die Universalität allerdings einmal sichergestellt, dann müssen wir den Aktivitätsträger, d.h. die Hardware zumindest aus Sicht der klassischen Berechenbarkeitstheorie nicht länger beachten. Programme sind gewissermaßen *reine* Maschinen, ohne den Ballast und die Unschärfen mechanischer oder physikalischer Bedingungen. Auf dieser Ebene gibt es keine Alterungs- oder Abnutzungsfehler mehr, sondern nur noch Entwurfsfehler durch den Entwickler. Der Programmierer konzentriert sich auf die zweite Maschine – die Softwaremaschine – die er als Zeichenkette konstruiert. Programme sind damit einerseits lineare Zeichenketten, die nach starren Regeln aus einem gegebenen Alphabet von Zeichen aufgebaut sind, gleichzeitig bedeuten die Zeichen Handlungsanweisungen für einen realen Mechanismus. Da alle denkbaren Maschinen vollständig auf Zeichenebene beschrieben werden können, fallen Alphabetisierung und Mechanisierung in der Theorie der Berechenbarkeit in eins.

- *Gleichbehandlung von Operanden und Operatoren*
  Das Konzept der Speicherprogrammierung erlaubt es, das Programm zusammen mit den zu verarbeitenden Daten in den Speicher der Maschine einzulesen und es anschließend auszuführen. Entscheidend ist nun, dass dabei für die Maschine der Unterschied zwischen Programmanweisungen (Operatoren) und Daten (Operanden) aufgehoben wird. Daten und Befehle werden nicht nur im selben Zeichenalphabet codiert, sondern auch im selben Speicher abgelegt. Durch dieses Prinzip werden überhaupt erst Übersetzungsprogramme wie Compiler und Interpreter möglich. Ein Compiler ist in dieser Betrachtungsweise ein Programm, das Daten manipuliert, die anschließend Befehle darstellen. Die Gleichbehandlung von Daten und Programmen ermöglicht auch, dass

56 Siehe hierzu Zwischenspiel: Universelle Turingmaschinen & minimale Kontrollstrukturen.

Programme sich während der Verarbeitung selbst lesen und modifizieren. Wichtig ist an dieser Stelle, dass das Prinzip der Universalität nicht voraussetzt, dass die Maschine in der Lage ist, die Befehle eines Programms zu manipulieren. Es reicht aus, wenn eine datenabhängige Ausführung der Befehlsreihenfolge gewährleistet wird, d.h. wenn erst während der Verarbeitung entschieden werden kann, welche Befehle mit welcher Häufigkeit ausgeführt werden. Die universelle Turingmaschine ist in der Lage, jede andere Maschine zu simulieren, indem das Programm auf einen bestimmten Bereich des Bandes geschrieben wird und die Daten auf einen anderen. Die universelle Maschine wird dann die Instruktionen im Befehlsbereich des Bandes lesen und gemäß der dort stehenden Anweisungen den Datenbereich manipulieren. Universalität kann gewährleistet werden, ohne dass die Maschine den Befehlsbereich des Bandes verändert.

Die klassische programmierbare Maschine, wie wir sie eben charakterisiert haben, ist zwar eine sehr trickreiche und mächtige Vorrichtung, aber insgesamt noch im relativ engen Rahmen des rationalen, mechanistischen Denkens verortet. Diese klassische Maschine ist gedacht als reine Rechen- und Zeichenmanipulationsmaschine. Sie ist selbst beschrieben als Zeichenkette und erhält als Eingabe ebenfalls Zeichen, manipuliert diese auf der Basis sequentieller, vollkommen deterministischer Verarbeitungsschritte und gibt wieder Zeichen aus. Der klassische Algorithmenbegriff sieht auch keine Interaktion zwischen Benutzer und Programm vor. Programm und Daten stehen zu Beginn der Berechnung vollständig im Speicher, wobei die Berechnung selbst nach einer endlichen Anzahl von Schritten stoppen soll. Von der Berechnung selbst werden keine neuen Qualitäten erwartet. Im Zentrum der klassischen Berechenbarkeitstheorie stehen klar spezifizierbare Aufgaben, die durch eine endliche Folge von Handlungsanweisungen gelöst werden können. Die Handlungsanweisungen stehen vor Beginn der Berechnung fest und werden im Verlauf der Berechnung auch nicht mehr verändert. Es macht dabei auch keinen Unterschied, ob die Anweisungen mit Papier und Bleistift durchgeführt werden oder auf einem Computer. Der Begriff der 'Zeit' existiert in der klassischen Algorithmentheorie nur implizit durch die Anzahl der notwendigen Berechnungsschritte. Wie lange die Ausführung der einzelnen Schritte tatsächlich braucht, ist theoretisch belanglos. Zeit- und Speicherplatzanforderungen werden erst in der jüngeren Komplexitätstheorie untersucht.

Der Rest dieses ersten Kapitels soll darauf verwendet werden, deutlich zu machen, dass das klassische Bild der Rechen- und Zeichenmanipulationsmaschine, wie es in der ersten Hälfte des zwanzigsten Jahrhunderts entwickelt wurde, der heutigen Praxis mit ihren vielfältigen Anwendungen und sehr weit reichenden Zielsetzungen, insbesondere im Feld der Medien, nicht mehr gerecht wird. Als Resümee lässt sich festhalten, dass die klassische Rechen- und Zeichenmanipulationsmaschine, die ausschließlich Zeichen verarbeitet und auch selbst nur aus Zeichen besteht, ein Produkt der rationalistischen Tradition und der von

ihr ausgehenden Formalisierung unserer Weltbilder ist. Die mathematischen Beschreibungen galten zu Beginn dieses Entwicklungsprozesses als objektive Wahrheiten über Bereiche der Realität. Im Laufe der philosophischen und wissenschaftlichen Ausdifferenzierung wurden sie zunehmend zu Modellen eines zu untersuchenden Originals. Die Forschungspraxis ist sich heute bewusst, dass sie mit ihren formalen Systemen immer nur Teilsysteme und Vereinfachungen des Ganzen betrachtet. Wir erfassen durch unsere Modelle die Wirklichkeit nie vollständig, sondern immer nur in Teilaspekten und der Zusammenhang zwischen formalem Modell und Original wird dabei in der Regel als Homomorphismus erklärt, d.h. als strukturerhaltende Abbildung, die von Seiten des formalen Modells defizitär ist. Das Modell erfasst die Wirklichkeit also nur selektiv, d.h. auf einer bestimmten Ebene und aus einer bestimmten Perspektive. Das Original bleibt dabei weitestgehend eine *Black Box*, die nicht offen legt, durch welche spezifischen Ursachen und Verbindungen sie ihr Verhalten generiert. Unsere solchermaßen eingeschränkten formalen Modelle sind aber – und das ist ihre große Stärke – Handlungstheorien, d.h. sie sind operationalisierbar und damit auf jedem Rechner ausführbar. An diesem Punkt vollzieht sich nun die entscheidende Wendung. Nachdem billige und leistungsfähige Universalhardware überall zur Verfügung steht und wir ausreichend Erfahrung mit unseren Modellierungen gesammelt haben, lösen wir sie aus dem Realität-Abbild-Verhältnis. Programme, als Realisierungen formaler Systeme, modellieren zunehmend nicht mehr nur Teilaspekte der Wirklichkeit, sondern erzeugen ihre eigenen Wirklichkeiten. Computer entwickeln sich verstärkt vom Werkzeug der Analyse zum Mittel der Synthese. Die Maschine begnügt sich nicht länger mit der Beschreibung der Wirklichkeit mit Hilfe von Symbolen, sondern greift vermittels algorithmischer Symbole zunehmend in die Konstruktion der Wirklichkeit ein.

1.3 Die hybride Maschine

„Damit ist der Rückschritt in die Abstraktion (und die Kulturgeschichte als fortschreitende Abstraktion) beendet. Zahlen sind nulldimensional, und aus der Null kann nichts mehr abgezogen werden. Daher: Zuerst Imagination, dann Konzeption, dann Kalkulation (zuerst Bild, dann Text, dann Algorithmus), und damit hat die Entfremdung ihren Höhepunkt erreicht, und weiter geht es nicht mehr. Das ist der Punkt, von welchem ab sich die Kulturgeschichte in die entgegengesetzte Richtung wendet."

Vilém Flusser[1]

In diesem Abschnitt werden neuere Entwicklungen des Computers beleuchtet, die vor allem mit den Forschungen der zurückliegenden 50 Jahre verbunden sind. Computeranwendungen beschränken sich heute längst nicht mehr auf rein numerische oder zeichenbasierte, ausschließlich die Ratio ansprechende Interaktionen mit dem Benutzer, sondern bedienen sich multimodaler Interfaces, die physiologisch und kulturell determinierte Wahrnehmungsprozesse einbeziehen. Die komplexen Anwendungen erstrecken sich zudem auf alle Bereiche des wirtschaftlichen, wissenschaftlichen und kulturellen Alltags. Durch die enorme Ausweitung der Methoden und Praktiken ist die in den letzten Jahrzehnten entstandene hybride Maschine nicht mehr auf der Basis der formalen, in der Kognition verhafteten Tradition verstehbar. Begriffe wie flottierende Codes, Selbstreferenz, Metaarchitektur, Interface und Heterarchie sollen helfen, sich in der neuen Praxis der Maschine zu orientieren. Eine Theorie der hybriden Maschine steht allerdings noch aus.

Zur Charakterisierung der gegenwärtigen Computerpraxis verwenden wir den Ausdruck hybrid (lat. gemischt, von zweierlei Herkunft), um die veränderten Verhältnisse von Anwendungen der klassischen Rechenmaschine abzugrenzen. Der Begriff des Hybriden scheint aus verschiedenen Gründen geeignet, die neue Situation zu beschreiben. Hybrid ist die neue Maschine, weil der rationale, zeichenbasierte Zugang zunehmend in den Hintergrund tritt und mit den immer komplexer werdender Interfaces akustische, visuelle und taktile Zugänge immer wichtiger werden. Neben den bisher dominierenden rationalen Fähigkeiten werden Wissen über Wahrnehmungsvorgänge und Wahrnehmungstheorien für den Programmierer damit zunehmend wichtig. Computeranwendungen, die im Kern nicht Rechenaufgaben lösen sollen, sondern mit Bildern und Klängen umgehen, können nicht mehr losgelöst von den Interfaces verstanden werden. Die hybride Maschine besteht immer aus dem digitalen Rechner und dem analogen Interface. Zum Zweiten soll der Begriff hybrid deutlich machen, dass der Rechner nicht mehr als isolierte Einheit in Erscheinung tritt, sondern als Knoten in komplexen Netzwerken seine Funktionen erfüllt. Das Internet, Funknetze

1 Die technische Imagination, in: Kunsthochschule für Medien (Hrsg.), Lab – Jahrbuch 1999 für Künste und Apparate, S. 344 f.

oder die neuen Handynetze sind einige Beispiele für die Vielzahl heterogener, miteinander verbundener Netzwerke aus rechnenden und kommunizierenden Prozessen und Maschinen. Zum Dritten ist die neue Maschine hybrid, weil komplexe Arbeitsprozesse am Computer heute in der Regel symbiotisch sind. Wir bearbeiten Probleme, die nur noch Mensch und Maschine im interaktiven Zusammenspiel angehen können, kein Teilsystem – weder der Mensch noch die Maschine – könnte diese alleine bewältigen.[2] Schließlich und endlich soll der Begriff hybrid auch noch in seiner griechischen Bedeutung (griech.: hybrid = hochmütig, überheblich, vermessen) verwendet werden. Zwischen den großen Leitbildern der Informatik sowie daraus sich ableitenden konkreten wirtschaftlichen Zielsetzungen und der Beherrschbarkeit und Überschaubarkeit der Folgen haben sich riesige Abgründe aufgetan. Die Selbstkontrolle ökonomischer Strukturen ist inzwischen offensichtlich soweit zurückgegangen, dass Software im vollen Wissen ihrer Unzulänglichkeit und Fehlerhaftigkeit auf den Markt gebracht werden kann. Alleine das sollte genügen, diese Technologien viel strikter dem Prinzip Verantwortung zu unterstellen. Nach Jahrzehnten des Software Engineering liegt aber auch die Vermutung nahe, dass diese Systeme prinzipiell nicht vollständig beherrschbar oder in ihrem Verhalten detailliert vorhersagbar sind. Wir werden versuchen, im Folgenden einige Hinweise darauf zu geben. So wenig uns die Versicherung des Physikers nützt, dass es sich beim Ziehen der Lottozahlen um einen physikalisch determinierten Vorgang handelt, den man theoretisch exakt beschreiben kann, hilft uns ab einer gewissen Komplexität der Computeranwendung die Beschwichtigung des Informatikers, dass Computer deterministische Maschinen sind.

1.3.1 Code-Reflexivitäten

„Münchhausens Theorem, Pferd, Sumpf und Schopf, ist bezaubernd, aber vergiss nicht: Münchhausen war ein Lügner."
aus Hommage à Gödel, von Hans Magnus Enzensberger[3]

Bei Kant finden wir in seiner *Kritik der Urteilskraft* den Gedanken, dass ein Organismus deshalb kein Automat sein kann, weil er sich nicht fortpflanzen kann. Ein Organismus sei immer organisierend und selbstorganisierend, ein Automat dagegen bloß organisierend. Für heutige Automaten und Maschinen trifft diese Feststellung längst nicht mehr zu. Spätestens durch die Arbeiten John von Neumanns über Strukturen der Selbstreproduktion kennen wir Automaten, die sich nicht nur selbst reproduzieren, sondern die Automaten bauen, die komplizierter sein können als sie selbst.[4] Da in der programmierbaren Maschine sowohl Daten

2 Siehe hierzu auch Zwischenspiel: Interfaces – Von Menschen und Mäusen, insbesondere den Abschnitt zu Douglas Engelbart.

3 In: Die Elixiere der Wissenschaft, S. 9.

4 Siehe Zwischenspiel: Zelluläre Automaten & Selbstreproduktion.

als auch Programme mit den gleichen Codes beschrieben werden und im gleichen Speicher liegen, können die Zeichen in neue Verhältnisse zueinander treten. Wir werden in diesem Abschnitt zunächst kurz zeigen, dass sich selbst beobachtende und verändernde Systeme für die gesamte neuere Kybernetik – die als Kybernetik 2-ter Ordnung in der Literatur bekannt ist – von großer Bedeutung sind. Aber auch eine ganze Reihe praktischer Anwendungen, hier insbesondere Lernverfahren und 'genetische Programmierung' stützen sich auf komplexe Beziehungen der Zeichen zu sich selbst. Durch die Gleichsetzung von Daten und Programm kann die Frage: „Was ist Operator und was ist Operand?" nur noch standpunktabhängig beantwortet werden. Vor dem Hintergrund des Wechselspiels zwischen Operatoren und Operanden erweisen sich viele scheinbar sehr unterschiedliche Methoden (Compiler, Viren, Evolutionsstrategien, Lernverfahren, selbstreproduzierende Systeme, etc.) als ähnlich und auf demselben Grundprinzip beruhend. Neben Code-Reflexivität ist diesen Strategien gemein, dass sie versuchen, ihren eigenen formalen Rahmen zu sprengen und neue Qualitäten hervorzubringen, die vom Programmierer nicht explizit vorgegeben werden. Nicht nur die Daten (Operanden) werden dabei einer Verarbeitung unterzogen, sondern auch die Programme (Operatoren), d.h. Programmcodes schreiben und verändern Programmcodes. Wir können die Vorgehensweisen als Münchhausenstrategien bezeichnen, da die Programme gewissermaßen versuchen, sich am eigenen Schopf aus dem Sumpf ihres Unvermögens zu ziehen.

Abbildung 26:
Malcolm Fowlers sich selbst nagelnder Hammer als Metapher für Strategien der Reflexion und Selbstmodifikation.

1.3.1.1 Selbstreferenz und Kybernetik 2-ter Ordnung

„Ich schlage vor, die Kybernetik von beobachteten Systemen als Kybernetik erster Ordnung zu betrachten; die Kybernetik zweiter Ordnung ist dagegen die Kybernetik von beobachtenden Systemen."
Heinz v. Foerster[5]

Ein wichtiger Grundsatz klassischer Wissenschaft lautet: „Die Eigenschaften des Beobachters dürfen nicht in die Beschreibung seiner Beobachtung eingehen."[6] Objektivität kann nur garantiert werden, indem der Beobachter sich nicht mit in seine eigene Beobachtung einbezieht. Der Beobachter soll nicht Teil des Systems

5 Kybernetik der Kybernetik, in: KybernEthik, S. 89.
6 Ebd., S. 88.

sein, das er beschreibt. Diese Arbeitsmaxime lieferte – gleichviel ob sie prinzipiell eingehalten werden kann oder nicht – sehr weitreichende und praktisch wertvolle wissenschaftliche Ergebnisse. Auch die bekannten Feedback-Systeme sind in dieser Hinsicht klassische Systeme und vollkommen ungefährlich. Der Selbstbezug, der hier am Werk ist, ist ein rein iterativer Selbstbezug, der keinerlei Widersprüche in sich trägt. Wir wollen uns nun Systemen zuwenden, die gefährlicher sind. Für bestimmte Systeme ist die Forderung, den Beobachter nicht in die Systembeschreibung einzubeziehen problematisch und nicht aufrechtzuerhalten. Zurecht stellt Heinz v. Foerster fest, dass die Gesetze der Physik der Mensch schreibt, die Gesetze der Biologie sich dagegen selbst schreiben müssen.[7] Wenn wir, als biologische Systeme, grundsätzliche Aussagen über biologische Systeme machen wollen, können wir uns nicht mehr aus der Beschreibung unseres Beobachtungsgegenstandes herauszuhalten. Dieses Problem haben alle Systeme, die antreten, Aussagen über sich selbst zu machen, gleichviel ob es sich um biologische, formale oder technische Systeme handelt. Wir haben es in diesen Fällen mit einer besonderen Form des Feedbacks zu tun, der logischen Selbstreferenz. Popularität hat die Forderung, dass die Eigenschaften des Beobachters mit in die Beobachtung und nicht zuletzt die Bilanz der Beobachtung einbezogen werden müssen, durch die Theorien des so genannten 'radikalen Konstruktivismus' erlangt. Ein zentraler Begriff des radikalen Konstruktivismus ist der von Humberto Maturana eingeführte Begriff der Autopoiese (griech.: autos = selbst, poiein = machen). Mithilfe der Autopoiese soll vor allem die Selbstorganisation bei Lebewesen erklärbar werden. Der Begriff wird von Maturana wie folgt definiert:[8] „Die autopoietische Organisation wird als eine Einheit definiert durch ein Netzwerk der Produktion von Bestandteilen, die 1. rekursiv an demselben Netzwerk der Produktion von Bestandteilen mitwirken, das auch diese Bestandteile produziert, und 2. das Netzwerk der Produktion als eine Einheit in dem Raum verwirklichen, in dem die Bestandteile sich befinden." Die logischen Probleme im Zusammenhang mit selbstreferentiellen Aussagen sind in der Philosophie seit der Antike bekannt. Mit der Arbeit des Logikers Kurt Gödel wurden Selbstbezüglichkeiten ins Zentrum der Computertheorie gerückt. Wesentliche Erkenntnisse über die Grenzen der Berechenbarkeit beruhen auf der Tatsache, dass hinreichend komplexe Systeme (lebende Organismen, aber auch Maschinen) zur Selbstreferenz fähig sind. Wenn Systeme Aussagen über sich selbst, ihre Eigenschaften und Verhalten machen, können sie sich in unauflösbare Widersprüche verstricken. Solche Phänomene heißen in der Literatur 'Antinomien' bzw. 'Paradoxien'. Insbesondere im deutschen Sprachgebrauch wird – im Gegensatz zum Englischen – scharf zwischen einer Antinomie und einer Paradoxie unterschieden. Während eine Antinomie ein echter, unauflösbarer Widerspruch ist (eine Aussage und ihr Gegenteil sind gleichzeitig wahr), ist eine Paradoxie ein scheinbarer Widerspruch, der nur dem Anfänger als solcher erscheint. Des

7 V. Foerster lehnt den Anspruch auf Objektivität allerdings generell ab, nicht nur für biologische Systeme.

8 In: Erkennen – Die Organisation und Verkörperung von Wirklichkeit, S. 158.

weiteren wird in der Literatur eine Unterscheidung zwischen semantischen und syntaktischen Antinomien getroffen. Semantische Antinomien entstehen durch missbräuchliche Verwendung der Umgangssprache, wobei die Bedeutung eine wesentliche Rolle spielt. Syntaktische Antinomien sind dagegen Widersprüche, die innerhalb eines formalen logischen Systems auftreten. Die drei Kennzeichen semantischer Antinomien sind:

- Selbstbezogenheit:
  „Dieser Satz enthält fünf Wörter." Ist selbstbezogen, aber weder widersprüchlich noch zirkelhaft.
- Widersprüchlichkeit:
  „Dieser Satz enthält sechs Wörter." Ist selbstbezogen und widersprüchlich, aber nicht zirkelhaft.
- Zirkelhaftigkeit:
  „Dieser Satz ist falsch." Ist selbstbezogen, widersprüchlich und zirkelhaft.

Eine klassische semantische Antinomie lautet: „Epimenides der Kreter sagt, dass alle Kreter Lügner sind." Semantische Antinomien, die umgangsprachliche Konstruktionen sind, erweisen sich allerdings keineswegs als harmloser als syntaktische Antinomien. Die Arbeiten von Gödel haben gezeigt, wie man semantische Antinomien in formalen Systemen simulieren kann. Die algorithmische Betrachtung der Antinomie zeigt jedenfalls eindeutig, dass den introspektiven Fähigkeiten von Algorithmen Grenzen gesetzt sind. Innerhalb eines Programms (Formalismus), sind bestimmte Aussagen über dieses Programm nicht möglich. Ein Beispiel ist die Unentscheidbarkeit des Halteproblems.[9] Andererseits gibt es eine Vielzahl selbstbezüglicher Anwendungen, die aus praktischer Sicht sowohl sinnvoll sind als auch ungefährlich hinsichtlich logischer Verstrickungen. Solange nur Selbstbezogenheit realisiert wird und Widersprüchlichkeit sowie Zirkelhaftigekeit vermieden werden können, lassen sich auf dieser Basis leistungsfähige Anwendungen konstruieren.

1.3.1.2 Computational Reflection – maschinelle Selbstbetrachtungen

„Ein Metalog ist ein Gespräch über ein problematisches Thema. In diesem Gespräch sollten die Teilnehmer nicht nur das Problem diskutieren, sondern die Struktur des Gesprächs als ganzes sollte auch für eben dieses Thema relevant sein."
Gregory Bateson[10]

Im engeren Sinn können wir Programmiersprachen erst dann wirklich als Sprachen bezeichnen, wenn sie in der Lage sind, Aussagen über ihre eigene Struktur zu machen. Dies unterscheidet den Begriff der Sprache von der allgemeinen Idee der Kommunikation. Sprache ist ein besonderes Kommunikationssystem, das über

9 Siehe Zwischenspiel: Algorithmisches Denken.

10 In: Ökologie des Geistes, S. 31.

sich selbst sprechen kann. „Die Bienen verfügen – nach meiner Definition – über keine Sprache. Sie können einander wohl zutänzeln. ... sie können nicht über ihr Vokabular sprechen, sie können sich auch nicht über ihre Grammatik unterhalten, ihr Kommunikationssystem selbst kann nicht innerhalb ihres Kommunikationsrepertoires kommuniziert werden. Die Sprache beginnt für mich aber dort, wo die Kommunikation einen Begriff von Kommunikation entwickelt und reflexiv wird.“[11]

In den 1980-er Jahren gab es unter dem Begriff ‘Computational Reflection’ ernsthafte Bemühungen, reflexive, d.h. selbstbezügliche Konstrukte auch in höheren Programmiersprachen zur Verfügung zu stellen. ‘Reflexivität’ bezeichnet in der Logik und Mathematik eine bestimmte Eigenschaft algebraischer Strukturen. Eine Relation heißt reflexiv, wenn Aussagen über die Beziehung zwischen Elementen der Struktur auch für jedes einzelne Element gelten, das Element also Aussagen über sich selbst macht. Für eine Menge von Zahlen ist die Relation „=“ (Identität) reflexiv, da jede Zahl mit sich selbst gleich ist. Für Mengen von Geraden ist die Relation „||“ (Parallelität) reflexiv, da jede Linie parallel zu sich selbst verläuft. Für Mengen allgemein ist die Teilmengenrelation reflexiv, da jede Menge Teilmenge ihrer selbst ist usw. In der Philosophie und Psychologie bedeutet ‘Reflexion’ die Fähigkeit des Menschen, sich selbst zu beobachten und darüber das eigene Verhalten – in gewissen Grenzen – zu kontrollieren und nachhaltig zu verändern. Und schließlich kennen wir den Begriff des Reflexes, der unter anderem von René Descartes benutzt wurde, um die Fähigkeit von Lebewesen zu bezeichnen, auf Einflüsse aus der Umwelt geeignet zu reagieren. Alle diese Bedeutungen spielen mit, wenn wir über Reflexion im Zusammenhang mit Computersystemen sprechen (Computational Reflection). Algorithmische Reflexionen stehen damit nicht nur im Kern der Berechenbarkeitstheorie (Antinomien, Cantorscher Diagonalisierungsbeweis, Unentscheidbarkeit, Halteproblem, etc.), sondern auch im Kern jeder Lern- und Evolutionstheorie. Eng verbunden mit den Problemen der algorithmischen Reflexion ist das Konzept der Metaarchitekturen, d.h. die Verwendung hierarchisch höher stehender Systeme, die in der Lage sind, Auskunft über untergeordnete Subsysteme zu geben. Der Katalog der Kataloge ist ein Beispiel für ein solches Metasystem.

Eine wesentliche Fähigkeit des Menschen, die besonders häufig angeführt wird, wenn es gilt, die Differenz zwischen Mensch und Tier sicher zu stellen, ist die Reflexion. Menschen sind fähig, sich selbst zu beobachten und ihr eigenes Denken und Handeln zu reflektieren. Im Bereich der künstlichen Intelligenz hat diese Fähigkeit erstaunlicherweise bisher nur relativ wenig Beachtung gefunden. Unter ‘Computational Reflection’ werden alle Aktivitäten zusammengefasst, die ein System durchführt, wenn es Berechnungen über seine eigenen Berechnungen anstellt. Ohne dies explizit unter dem Begriff der Reflexion durchzuführen, stellen

11 Heinz von Foerster, Der Anfang von Himmel und Erde hat keinen Namen, S. 162.

Computer schon immer nicht nur Berechnungen über den jeweiligen Problem- und Aufgabenbereich an, sondern führen auch selbstbezogene Berechnungen durch. Beispiele sind: Performance-Statistiken über die Auslastung des Rechners und seiner Ressourcen, Sammlung von Informationen über den Debugging-Prozess bei Compilern, Selbstmodifikationen innerhalb von Lernverfahren oder Berechnungen darüber, welche Befehle innerhalb eines laufenden, nicht-deterministschen Programms als nächstes ausgeführt werden sollen. In den meisten Fällen werden diese Berechnungen eher in 'ad hoc Manier' gelöst, als auf der Basis systematischer Mechanismen, die bereits auf Programmiersprachen- oder Betriebssystemebene angeboten werden. In einigen Programmiersprachen werden aber auch explizite Mechanismen bereit gestellt, um reflexive Berechnungen zu unterstützen. Bekannte Beispiele sind die Programmiersprachen FORTH, COMMON LISP, SCHEME und SELF.

In Anlehnung an Pattie Maes[12] sollen hier lediglich einige grundlegende Begriffe skizziert werden, die für das Verständnis des Ansatzes wichtig sind:

- Ein Rechensystem *stellt Berechnungen* über einen gewissen Bereich seiner Außenwelt – den Anwendungsbereich – *an*, wenn es Daten vorhält und manipuliert, die diesen Teil der Welt repräsentieren.
- Ein Rechensystem *handelt* innerhalb eines Teils der Welt, wenn die Daten, die diesen Ausschnitt der Welt repräsentieren, mit diesem 'kausal verbunden' sind, d.h. dass sich die Änderung des einen direkt auf den anderen auswirkt. Die Verbindung zwischen inneren Berechnungen und äußeren Handlungen ist durch das Interface realisiert.

Die nachfolgenden Abbildungen zeigen die Grundstruktur unterschiedlicher Systeme. Rechensysteme sind Systeme, die Berechnungen über einen bestimmten Bereich ihrer Außenwelt anstellen. Dieses Gebiet der Außenwelt, der so genannte Anwendungsbereich (Gegenstandsbereich, Referenzbereich), wird in Form von Daten (numerische und alphabetische Zeichen) im System repräsentiert. Die Befehle des Programms beschreiben, wie die Daten manipuliert werden. In der objektorientierten Programmierung werden zusammengehörige Teile von Daten und Befehlen zu so genannten Objekten zusammengefasst. Da sowohl Daten als auch Programm aus Zeichen bestehen, braucht es noch einen Aktivitätsträger – den Prozessor – der die Befehle des Programms auf die Daten anwendet, also die eigentliche Berechnung durchführt. Wir wollen hierbei explizit zwischen 'Berechnungen' und 'Handlungen' unterscheiden. Berechnungen über die Außenwelt führen nicht direkt zu 'Zustandsänderungen' in der Außenwelt, während man nur von Handlungen sprechen kann, wenn die Berechnungen auf direkte Weise Änderungen in der Umgebung des Systems bewirken. Ein Roboter führt Handlungen aus, während zum Beispiel ein Statistikprogramm oder die simulationsbasierte Wettervorhersage lediglich Berechnungen durchführen. Um

12 Vgl.: Computational Reflection, S. 34.

von Handlungen sprechen zu können, ist also eine kausale Verbindung zwischen Daten über die Außenwelt und der Außenwelt selbst erforderlich. Änderungen bestimmter Daten bewirken direkt Änderungen im entsprechenden Bereich der Außenwelt.

Die Frage lautet, was wir in diesem Zusammenhang unter einem 'reflexiven System' verstehen wollen. Ein Rechensystem heißt 'reflexiv', wenn es kausal verbundene Daten über sich selbst enthält. Eine Programmiersprache für reflexives Programmieren müsste demnach verschiedene Konstrukte anbieten. Der Interpreter dieser Sprache muss unter anderem zur Laufzeit des Programms Daten anbieten, die das System selbst repräsentieren. Das Programm hat dann

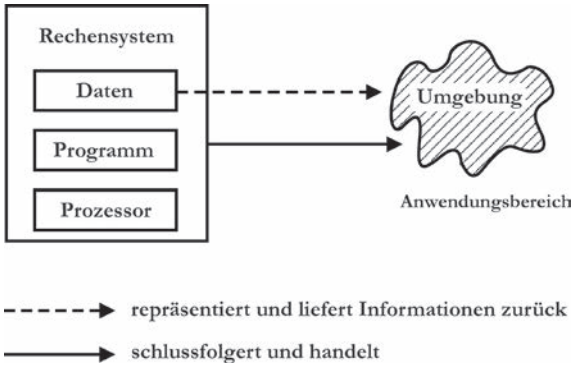


Abbildung 27: Schematische Darstellung eines Rechensystems.

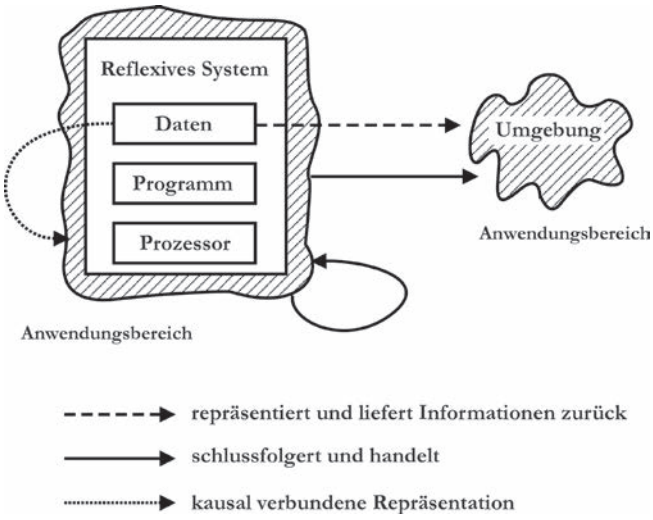


Abbildung 28: Schematische Darstellung eines reflexiven Systems.

die Möglichkeit, anhand dieser Daten verschiedene Berechnungen anzustellen, insbesondere kann es diese Daten und damit seine eigene Repräsentation direkt manipulieren. Das System muss in diesem Fall sicher stellen, dass die kausale Verbindung zwischen den Daten und dem, was sie repräsentieren, gewährleistet ist. Daraus folgt, dass die Modifikationen, die das Programm durchführt, von diesem auch reflektiert werden müssen.

1.3.1.3 Metasysteme

„Das Buch, das in der Welt am ersten verboten zu werden verdiente, wäre ein Katalogus von verbotenen Büchern."

Lichtenberg[13]

Bei Pattie Maes wird ein weiterer Typ von Systemen thematisiert, der aus praktischer Sicht mindestens ebenso wichtig ist wie selbstreferenzielle Systeme und der auch eng mit diesen verbunden ist – Metasysteme.

- Ein Rechensystem heißt *Metasystem*, wenn es Berechnungen oder Handlungen

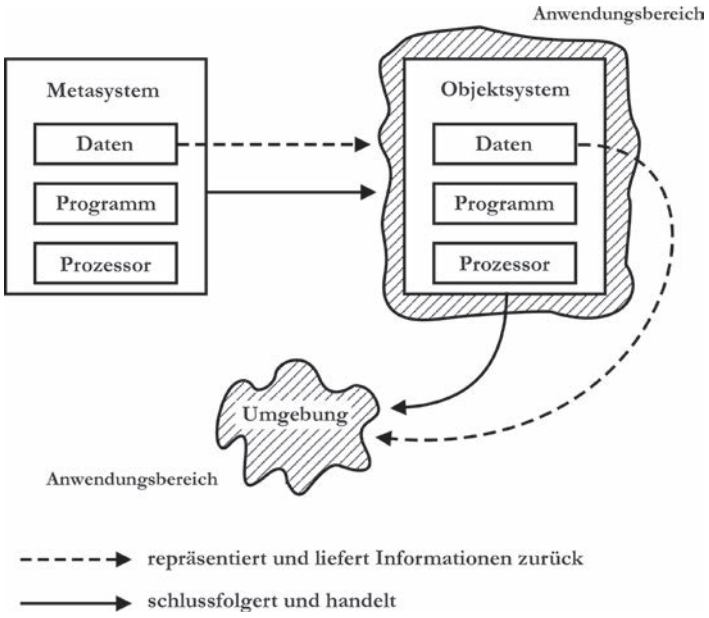


Abbildung 29:
Schematische Darstellung eines Metasystems.

13 In: Franz Mautner (Hrsg.), G.C. Lichtenberg Sudelbücher, S. 343.

über ein anderes Berechnungssystem anstellt. Dazu muss es Daten enthalten, die das andere Rechensystem repräsentieren.

Aus der Praxis allgemein bekannt ist der Begriff der Metadaten. Dies sind konsequenterweise Daten, die Informationen über die Struktur anderer Daten enthalten. Jedes Datenformat enthält solche Anteile, die statistische und strukturelle Aussagen über die Organisation der eigentlichen Daten machen. Das Wenige, das wir hier über reflexive Systeme und Metasysteme gesagt haben, macht bereits deutlich, dass die Grenzen zwischen diesen Systemen nicht immer klar gezogen werden können. Ob ich ein Gesamtsystem als reflexives System bezeichne oder ob ich dieses System in zwei Subsysteme unterteile, bestehend aus einem Rechensystem und einem Metasystem, das die Reflexion des Rechensystems durchführt, bleibt der eigenen Interpretation überlassen. Metasysteme und reflexive Systeme stellen Berechnungen über Bereiche an, die von der gleichen Struktur sind wie sie selbst, d.h. die selbst zeichenbasierte Rechensysteme sind. Der wesentliche Unterschied zwischen Metaarchitekturen und reflexiven Architekturen – und hier ist dann doch die Grenze zu ziehen – besteht in der Art der gegenseitigen Einflussnahme. Bei Metaarchitekturen wird das Basissystem durch das Metasystem manipuliert, das Metasystem selbst wird sich dabei nicht ändern, während in reflexiven Systemen der Verlauf der Berechnungen zumindest prinzipiell Einfluss auf alle Bereiche seiner eigenen Repräsentation nehmen kann. Das Metasystem hat gegenüber dem Basissystem einen erweiterten Horizont. Das starre Metasystem bildet den eigentlichen Rahmen, innerhalb dessen sich das Gesamtsystem bewegen kann. Es implementiert damit eine ganz bestimmte Sicht auf die Welt, bzw. den Ausschnitt der Welt, den das System modelliert und bestimmt damit auch die Grenzen für die Freiheiten des Systems.

Wir wollen zur Verdeutlichung der Vielfalt an Beziehungen, die zwischen den Zeichen als Operanden und den Zeichen als Operatoren herstellbar sind, einige Beispiele nennen.

Compiler & Bootstrapping

Compiler sind Programme, die als Eingabe Zeichen erhalten (nämlich das Programm a) und daraus neue Zeichen (Programm a') erzeugen. Wichtig an der Funktion des Compilers ist, dass er seine Arbeit in Schritt 1 so ausführt, dass in Schritt 2 ein Umschlag des Programms vom Status des Operanden in den Status des Operators stattfinden kann (das Umschlagen wird mit ⇓ bezeichnet).

Schritt 1:	Compiler (a) →	a'
		⇓
Schritt 2:		a'(Daten) → Ergebnisse

Interpreter dagegen sind Programme, die keine eigenständig lauffähigen Dateien erzeugen, sondern zeilenweise den Programmcode übersetzen und ihn direkt zur Ausführung durch den Prozessor bringen. Da sich jeder Compiler auf eine bestimmte Programmiersprache und eine bestimmte Maschinensprache bezieht, müssen für n Programmiersprachen und m Maschinensprachen n x m Compiler geschrieben werden, was einen mit der Anzahl n und m quadratisch wachsenden Aufwand bedeutet. Wichtig ist, dass Compiler letztlich als Maschinenprogramme existieren müssen, deren Erstellung eine sehr aufwendige Programmieraufgabe ist. Um diesen Aufwand zu reduzieren, wurden z.B. Zwischensprachen eingeführt,[14] bzw. auch so genannte Compiler-Compiler. Der Compiler-Compiler verlangt als Eingabe die syntaktische und semantische Beschreibung einer problemorientierten Sprache und kann daraus dann entweder direkt einen Compiler generieren, oder selbst als Interpreter fungieren und eingegebene Programme übersetzen. Eine weitere Möglichkeit ist die Verwendung von so genannten Implementierungssprachen, das sind Sprachen, die sich besonders für den Compilerbau eignen. Compiler werden hier also selbst wieder in einer höheren Programmiersprache geschrieben. Wir sehen bereits, dass Compilerbau ein hochgradig selbstbezügliches System ist. Durch geschickte Code-Reflexivitäten besteht deshalb auch die Möglichkeit, Compiler bis zu einem gewissen Grad durch sich selbst zu übersetzen. Bei Schneider[15] wird ein 'Bootstrapping Verfahren' beschrieben, das bei Vorliegen eines Compilers a, für eine bestimmte Implementierungssprache und eine Rechenanlage A, mit geringem Aufwand einen Compiler b für diese Implementierungssprache und eine Maschine B erstellt.

1. Im Compiler a, der in der Implementierungssprache selbst vorliegt, werden an allen Stellen, an denen Maschinensprache für die Maschine A erzeugt wird, die entsprechenden Befehle der Maschine B eingesetzt. Durch diesen Vorgang erhält man a'.
2. Man übersetzt a' mit Hilfe von A und a in die Maschinensprache von A und erhält a".
3. a" ist ein lauffähiges Programm für die Rechenanlage A. Dieses Programm erhält als Input wiederum a' und erzeugt ein Maschinenprogramm b, das auf der Rechenanlage B lauffähig ist. B ist der gewünschte Compiler, der auf B lauffähig ist und Programme erzeugt, die ebenfalls auf b laufen.

Genetische Programmierung

Evolutionäre Algorithmen arbeiten auf Basis der Populationen von Individuen, die von Generation zu Generation den größten Teil ihres genetischen Codes vererben.[16] Die Eigenschaften des einzelnen Individuums sind als Genotypen

14 Siehe Kapitel 2, die Java Virtual Machine.

15 Vgl. Hans Jürgen Schneider, Compiler – Aufbau und Arbeitsweise, S. 26.

16 Siehe Zwischenspiel: Digitaler Darwinismus.

codiert, die das Erbgut eines Individuums repräsentieren und einen Bauplan für das Individuum liefern. Die Reproduktion einer Population von einer Generation in die nächste erfolgt durch identisches Kopieren, die Mutation einzelner zufällig gewählter Positionen des Codes, sowie den Austausch von Teilstrukturen zwischen zwei Individuen (Crossover). Veränderungen der Merkmale eines Individuums können ausschließlich am Genotypen – also den Zeichenketten vorgenommen werden, die Selektion der 'Besten' gemäß Fitness-Funktion erfolgt dagegen am Phänotypen. Der Phänotyp ist die physikalische Manifestation des Genotypen, d.h. die Merkmalsausprägung eines Individuums in seiner Umwelt. In der 'Genetischen Programmierung' werden spezielle Genotypen behandelt, nämlich Computerprogramme. Der Genotyp ist hier der Programmcode und der Phänotyp die Performance des Programms, wenn es auf einem Rechner ausgeführt wird. Auch erkennen wir wieder den Wechsel zwischen Operator und Operand. Für den genetischen Basisalgorithmus fungiert der Genotyp als Operand, der Phänotyp dieses Codes dagegen als Operator.

Selbstreproduktion

Das Prinzip der Selbstreproduktion nach John von Neumann arbeitet ebenfalls mit dem Wechselspiel von Operatoren und Operanden.[17] Der sich selbst reproduzierende Konstruktionsautomat D = A+B+C – bestehend aus einem allgemeinen Konstruktionsautomaten A, der anhand einer gegebenen Beschreibung F(X) jeden dazugehörigen Automaten X bauen kann, einem Kopierautomaten B, der Kopien von Beschreibungen anfertigt, und einem Steuerautomaten C, der den Produktionsablauf kontrolliert – fungiert im ersten Schritt als Operator. Er baut, wenn man ihm seine eigene Beschreibung F(D) eingibt, zunächst seine eigene Hardware nach. Im zweiten Schritt wird B eine Kopie der Beschreibung des Automaten D anfertigen. C wird dann diese Beschreibung in den neuen von D gebauten Automaten D' packen, diesen starten und sich von ihm abkoppeln. Hier findet der Wechsel statt. Während bis zum Zeitpunkt der Abkopplung vom gebauten Nachkommen, dieser als Operand für den bauenden Automaten fungiert hat, schlägt der Prozess nun um und der Operand macht sich selbst als Operator ans Werk, um neue Automaten zu bauen.

Viren

Ähnliche Mechanismen der Selbstreproduktion – allerdings ausschließlich auf Softwareebene und nicht auf Hard- und Softwareebene wie bei v. Neumann – verwenden Computerviren. Viren sind kleine Programme, die Sicherheitslücken in Betriebssystemen ausnutzen, um fremde Computer dazu zu bringen, Codes auszuführen, die aus Sicht der Besitzers dieser Maschinen zu unerwünschtem Verhalten führen, zum Beispiel Daten löschen oder den laufenden Betrieb

17 Siehe Zwischenspiel: Zelluläre Automaten & Selbstreproduktion.

stören. Viren sind deshalb tückisch, weil sie fremde Dateien, aber auch fremde Programme infizieren, d.h. sie schreiben sich in bestehende Codes ein. Die Verbreitung von Viren erfolgt in der Regel in einem mehrstufigen Verfahren. Zunächst untersucht der Rechner, von dem der Virus ausgeht, den Zielrechner nach Infektionsmöglichkeiten, um dann im Infektionsteil den Zielrechner anzuweisen, den eigentlichen Viruscode auf der eigenen Festplatte zu speichern und auszuführen. Die meisten Viren führen ihren Code nicht unmittelbar nach erfolgreicher Installation aus, sondern warten damit, bis bestimmte Bedingungen erfüllt sind. Wird der Virus gestartet, treten nicht nur die vom Hersteller des Virus beabsichtigten Störungen auf, sondern es wird gleichzeitig auch ein neuer Untersuchungsteil und Infektionsteil auf weiteren Rechnern gestartet. Auch hier sehen wir das inzwischen bekannte Muster des zeitlich alternierenden Fortschreibens des Codes zunächst als Operand, dann die Ausführung als Operator, der seine eigene Beschreibung wieder als Operand behandelt usw.

1.3.1.4 Heterarchien

„Wie Enzensberger in seiner Kritik des Orwellschen Mythos zeigt, kann man sich in der Tat kaum mehr vorstellen, dass es zu einem Megasystem zentralisierter Kontrolle kommen kann (ein Kontrollsystem des gegenwärtigen Telefonsystems müsste dieses in seiner Komplexität n-mal übersteigen, ist also praktisch ausgeschlossen). Aber es ist ein wenig naiv, zu glauben, die Zensur sei derart schon durch die Ausweitung der Medien liquidiert. Selbst auf lange Sicht bedeutet die Unmöglichkeit polizeilicher Megasysteme lediglich, dass die gegenwärtigen Systeme durch Feedback und Selbststeuerung in sich selbst diese nun nutzlosen Kontrollmegasysteme integrieren. Sie sind in der Lage, das was sie negiert, als zusätzliche Variable einzuführen. Sie sind in ihrer Operation selbst die Zensur: es bedarf keines Metasystems. Demnach hören sie nicht auf, totalitär zu sein: sie verwirklichen in gewisser Hinsicht das Ideal dessen, was man einen dezentralisierten Totalitarismus nennen könnte."

Jean Baudrillard[18]

Das bedeutendste und mächtigste Organisationsprinzip in Technik, Wissenschaft und Wirtschaft ist die Hierarchisierung von Systemen, d.h. die fortgesetzte Untergliederung ihrer Funktionalität in miteinander verbundene Subsysteme, die ihrerseits jeweils wieder hierarchisch strukturiert sind. Jede Ebene kontrolliert dabei die darunter liegende Ebene, die jeweils festgelegte Teilfunktionalitäten im Gesamtkontext erfüllt.[19] Bei der Entwicklung von Computern – sowohl der Hardware wie auch der Software – kommen in allen Phasen und in allen Bereichen hierarchische Organisationsprinzipien zum Einsatz. Nur durch Hierarchiebildung ist es möglich, überhaupt derartig komplexe Systeme zu bauen, die keine einzelne Person mehr über alle Ebenen hinweg versteht. Die heutige Praxis des Computers macht es unserer Ansicht nach erforderlich,

18 Requiem für die Medien, in: Kool Killer, S. 108.

19 Siehe auch Zwischenspiel: Komplexität – Abstraktionen, Module und Hierarchien.

dieses Organisationsprinzip durch das Prinzip der Heterarchien zu ergänzen. Gleichzeitig bildet der Begriff der Heterarchie den notwendigen – nach bisheriger Argumentation sich geradezu aufzwingenden – Schlussstein der Eigenschaften der hybriden Maschine. Wir werden im Folgenden insbesondere sehen, dass Hierarchie und Heterarchie nicht konkurrierende Organisationsprinzipien sind, sondern Heterarchie eine Erweiterung des enorm wichtigen hierarchischen Prinzips bedeutet. Beide Begriffe stammen vom griechischen 'archein' – der erste sein, herrschen – ab. Die Soziologie bezeichnet mit Hierarchie (von 'hieros' – heilig, zu den Göttern gehörig, priesterlich) Herrschaftssysteme von vertikal und horizontal fest gefügten und nach Über- und Unterordnung gegliederten Rängen. In der idealtypischen Hierarchie sind alle Entscheidungsbefugnisse, Kommunikations- und Informationswege, Kompetenzen und Verantwortlichkeiten pyramidenhaft aufgebaut.[20] Heterarchie (von 'heteros' – anders, verschieden) bezeichnet dagegen die Herrschaft vieler, verteilter Instanzen. Fälschlicherweise wird der Begriff Heterarchie häufig – insbesondere in der Gegenüberstellung mit der baumartigen Hierarchie – mit Netzartigkeit gleichgesetzt. Netzwerke sind allerdings nicht zwangsläufig Heterarchien, während umgekehrt aber alle Heterarchien netzwerkartig strukturiert sind. Netzartigkeit ist damit eine notwendige, aber keine hinreichende Bedingung für Heterarchie. Bemerkenswerter Weise wird der Terminus Heterarchie heute weniger in den Computerwissenschaften, als vielmehr in der Organisationstheorie diskutiert.[21] Hierarchische Strukturen scheinen den Anforderungen an Organisations-, Firmen- und Fabrikstrukturen des 21. Jahrhunderts nicht mehr gewachsen zu sein. Heterarchie wird deshalb als Idealform sich selbststeuernder, polyzentrischer Organisationen aufgefasst, die über die Fähigkeit verfügen, spontan und problemorientiert neue Strukturen auszubilden und diese wieder aufzulösen. Im Vergleich zu den baumartigen, hierarchischen Strukturen werden Heterarchien als besser geeignet für den Umgang mit Unsicherheit, Komplexität und Dynamik angesehen. Zum Beispiel sollen die Elemente einer Organisation zielgerichtet ein Netz aus flachen Hierarchien ausbilden, um schnell und effizient agieren zu können und sich nach Erreichen des Ziels wieder aufzulösen. Diese in den Wirtschaftswissenschaften entwickelten Vorstellungen zum Begriff der Heterarchie sollen uns als Vorbild für die Anwendung auf programmierbare Maschinen dienen.

Die Idee der heterarchischen Organisation geht zurück auf die Arbeiten des Neurophysiologen Warren McCulloch, der in einem 1945 publizierten, nur fünfseitigen Artikel „A Heterachy of Values Determined by the Topology

20 Vgl. 'Hierarchie' in: Meyers grosses Taschenlexikon, 5. Auflage, erscheinen im B.I. Taschenbuchverlag 1995.

21 Siehe zum Beispiel Markus Reihlen, Führung in Heterarchien, sowie Markus Schwaniger, Das Modell Lebensfähiger Systeme – Ein Strukturmodell für organisationale Intelligenz, Lebensfähigkeit und Entwicklung.

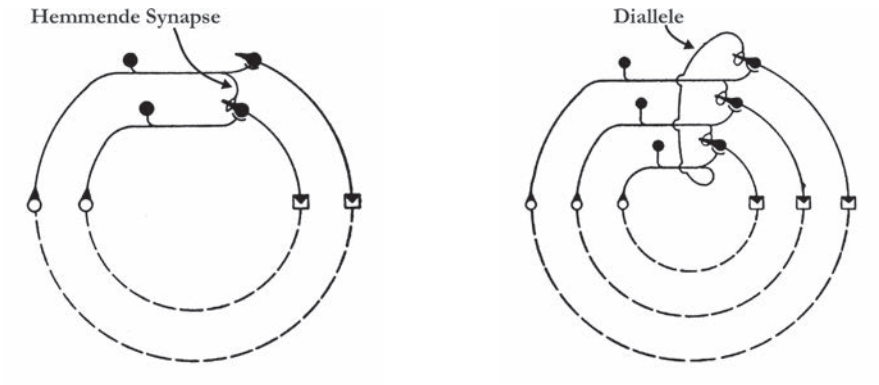


Abbildung 30/31:
Hierarchische Verknüpfung von Neuronen (links), Heterarchische Verknüpfung von Neuronen mit Diallele (rechts).

of Nervous Nets"[22] die Notwendigkeit nicht-transitiver Operationen zeigt, um neuronale Aktivitäten des Gehirns adäquat beschreiben zu können. McCulloch untersucht in diesem Artikel die topologische Struktur von miteinander wechselwirkenden Neuronen. Die Aktivitäten von Neuronen werden dabei als geschlossene Kreisbahnen behandelt, die sich aus einem in der Umwelt (Körper + Außenwelt) verlaufenden und einem in der Nervenzelle verlaufenden Teil des Weges zusammensetzen. Mit dem Begriff des 'Droms' wird dabei die zirkuläre Richtung der Ausbreitung eines Reizes von den Rezeptoren eines Nervensystems hin zu den Effektoren bezeichnet. Drome sind damit – vereinfacht ausgedrückt – gerichtete Rückkopplungsmechanismen, die in geschlossenen Kreisbahnen verlaufen, wobei ein Teil des Weges in der Umwelt zurückgelegt wird. Bei der Untersuchung von Zusammenschlüssen mehrerer Drome zu einem Netz, stellt McCulloch fest, dass bestimmte zielgerichtete Aktivitäten, bei denen zum Beispiel A den Vorzug vor B erhalten soll, B den Vorzug vor C, aber C den Vorzug vor A, nicht in flachen hierarchischen Strukturen realisiert werden können. Derartige Strukturen lassen sich nur mit Hilfe einer 'Diallele', einer die Fläche verlassende und übergreifende Verbindung, darstellen. Die einfachste Oberfläche, auf der solch ein Netz topologisch ohne Diallele abbilden lässt, ist der Torus.

In einem Brief an den Herausgeber der Zeitschrift, in der der Artikel erschienen ist, schlägt McCulloch ein weiteres Netz vor, das ebenfalls die Nicht-Transitivität der Operation abbilden kann, dabei aber nicht mit hemmenden Verbindungen arbeitet, sondern mit summativen. Gleichzeitig erklärt seine ergänzende Erläuterung sehr schön die Arbeitsweise des Netzes. McCulloch

22 Ursprünglich erschienen in: Bulletin of Mathematical Biophysics, Band 7, Chicago University Press, Chicago 1945, S. 89 – 93, erneut abgedruckt in: Embodiments of Mind, second Printing, MIT Press, Cambridge 1989, Deutsche Fassung: McCulloch, Warren S., Verkörperungen des Geistes, Springer-Verlag, Wien 2000, S. 41 – 46.

Abbildung 32 – 34:
Nicht-transitive Regeln für den Vorrang einzelner Schlagarten in Computerspielen. 'Leg sweep' schlägt 'forward kick' (links), 'forward kick' schlägt 'stomp' (mitte), 'stomp' schlägt 'leg sweep' (rechts).

schreibt im Brief an den Herausgeber: „Wenn drei Drome gegeben sind, von denen jedes über eine synaptische Verbindung reicht, die allein nicht genügt das nächste Neuron zu feuern, die aber von einem anderen Drom verstärkt werden kann, so ist ein Organismus, der nicht appetitiv auf einen von drei Sinnesreizen alleine reagiert, doch in der Lage, auf zwei zu reagieren, was wie eine Präferenz für einen der beiden Reize aussieht; drei solcher spezieller Wahlen können die Zirkularität der Werteanomalie aufzeigen. Man betrachte drei Drome – A, B und C –, die so verbunden sind, dass keines allein die Aktivität aufrechterhält, wenn nicht die afferente Komponente eines anderen Droms hinzukommt; das Netz sei so beschaffen, dass B (und nur B) notwendigerweise zu A beiträgt; entsprechend trage C (und nur C) zu B, und A (und nur A) zu C bei. Wenn ein einzelner Reiz a, b oder c dargeboten wird, gibt es keine Reaktion; wenn aber ein Reizpaar, a und b, oder b und c, oder c und a vorliegt, wird der Organismus entsprechend a, b oder c auswählen; und wenn a, b und c gegeben sind, kann der Organismus alle drei wählen."

Das Netz, das McCulloch hier beschreibt, ist topologisch äquivalent zu dem Netz, das er in seinem ursprünglichen Artikel beschreibt. Die Verbindungen zwischen den Neuronen sind jetzt lediglich nicht mehr hemmend (wie im Bild dargestellt), sondern summativ, d.h. damit eine Reaktion entsteht, sind Impulse aus zwei neuralen Endungen erforderlich, die sich addieren. Damit ist das Netz mit drei logischen UND-Gattern vergleichbar, jedes Gatter schaltet durch, wenn beide Eingänge belegt sind.

Bedeutsam an den Arbeiten von Warren McCulloch, die er teilweise zusammen mit Walter Pitts durchführt, ist die Entdeckung des Zusammenhangs zwischen formaler Logik und neuronalen Aktivitäten des Gehirns. Die formale Logik beruht bekannter Weise auf einem 'Alles-oder-nichts-Prinzip', d.h. eine Aussage ist entweder falsch oder sie ist wahr. McCulloch und Pitts finden überraschender Weise dieses Grundprinzip auch im Funktionsschema der Nerventätigkeit des menschlichen Gehirns. Wegen des 'Alles-oder-nichts'-Charakters der Nerventätigkeit können sie neuronale Ereignisse und Beziehungen zwischen ihnen mit den Mittel der Aussagenlogik beschreiben.[23] McCulloch fordert,

dass auch die formale und philosophische Logik sich dem Prinzip der Heterarchie öffnen und die bisher gültigen Wertehierarchien über Bord werfen müssen. „Aus der Sicht der Logik setzt man bei der Aufstellung einer Wertehierarchie voraus, dass Werte eine bestimmte Art von Größen sind. ... Experimentelle Ästhetik, Wirtschaftwissenschaft und die Wissenschaft der konditionierten Reflexe haben zu Fällen geführt, in denen die Präferenz unter gleich bleibenden Bedingungen zirkulär war. Ein solcher Fall wäre Grund genug gewesen, der Annahme, dass Werte irgendwie Größen sind, eine kategorische Absage zu erteilen. ... Zirkularitäten in der Präferenz zeigen nicht etwa Widersprüchlichkeiten an, sondern beweisen vielmehr Widerspruchsfreiheit einer höheren Ordnung, als sie unsere Philosophie sich je erträumen würde.“[24] Natürlich können solche Zirkelschlüsse – entgegen mancherorts vertretener Behauptung – auch mit klassischen logischen Operationen realisiert werden. Genauso leicht ist es, ein Programm anzuschreiben, das exakt die Arbeitsweise des McCulloch-Netzes nachbildet. Viel wichtiger ist die Erkenntnis, dass solche zirkelhaften Abhängigkeiten, die in der Mathematik als nicht-transitive Relationen bezeichnet werden, nicht logische Sonderfälle sind, die man mit klassischer Logik nachbilden kann, sondern, dass es sich hierbei um ein logisches Organisationsprinzip handelt, das ins Zentrum komplexer Systeme gestellt werden muss. Heterachische Ordnungsstrukturen finden sich nicht nur in der Neurophysiologie des Gehirns, sondern überall und insbesondere – so unsere Feststellung – zunehmend auch in der Computerpraxis. Ein sehr einfaches Beispiel, das exakt der zirkelhaften Ordnung McCullochs entspricht, ist das so genannte ‘game balancing’ in Computerspielen. Um die Ausgewogenheit des Spiels zu gewährleisten, darf es etwa in Kampfspielen keine Waffen geben, für die nicht auch eine Gegenwaffe existiert. In Zweikampfszenen gibt es keine dominanten Schläge, sondern für jede Aktion auch mindestens eine Gegenaktion, die diese neutralisiert. Diese zyklische Ordnungsstruktur ist die einfachste Form einer Heterarchie. Verallgemeinert wollen wir unter Heterarchien Organisationsstrukturen verstehen, die fähig sind, spontan problemorientierte Kooperationsformen auszubilden und diese nach Erreichen des Ziels auch wieder aufzulösen. Wichtig ist hierbei – und das gilt für Hierarchien gleichermaßen wie für Heterarchien –, dass sich diese Systeme mit ihren spezifischen Funktionen selbst erzeugen. Die Qualitäten komplexer Systeme sind immer Ergebnisse des Zusammenwirkens ihrer Komponenten, d.h. das Ganze ist mehr als die Summe seiner Teile.

Wie zu Beginn des Abschnitts deutlich gemacht, wollen wir den Begriff Heterarchie, d.h. die Herrschaft vieler verteilter Instanzen, vor allem für Systeme benutzen, in denen es mehrere gleichzeitig wirkende Aktivitätsträger gibt. Als einfache Veranschaulichungen elementarer Heterarchien in unserem Verständnis können die künstlerischen Arbeiten von Carlo Maria Mariani und von Maurits

23 Siehe Zwischenspiel: Logische Kalküle, Perzeptren und neuronale Netze.

24 Warren S. McCulloch, in: Verkörperungen des Geistes, S. 44.

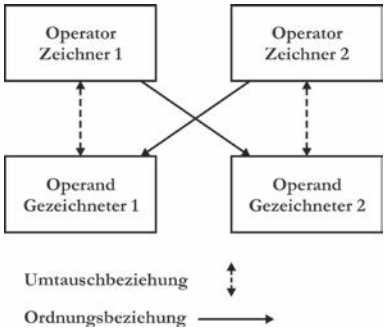


Abbildung 35/36:
Carlo M. Mariani (1983), *Die Hand unterwirft sich dem Intellekt* (links), Interpretation des Bildes als kanonische Heterarchie (rechts).

Cornelis Escher[25] dienen. Das gezeigte Bild von Mariani macht auch den engen Bezug zwischen Heterarchie und Selbstreferenz deutlich. Heterarchien sind gewissermaßen Selbstbezüglichkeiten, die über mehrere Orte verteilt sind. Beide Personen in Marianis Bild sind Zeichner und Gezeichnete zugleich; sie sind zur selben Zeit Operatoren und Operanden. Während Hierarchien gewissermaßen domestizierte netzartige Ordnungsstrukturen sind, sind echte Heterarchien sehr viel schwerer zu planen und zu realisieren. Organisationsprobleme, die wir hier unter dem Begriff Heterarchie besprechen, sind eng mit dem in der Informatik wichtigen und ausführlich behandelten Modell der Nebenläufigkeit (Parallelität) verbunden. Tatsächlich hat die Informatik eine Reihe mächtiger Werkzeuge entwickelt, die geeignet sind, eine Vielzahl von Problemen, die im Zusammenhang mit verteilten interagierenden Systemen auftreten, zu lösen. Als Beispiele für Modellierungsmethoden nebenläufiger Programmmodule seien hier die von Carl Adam Petri eingeführten und nach ihm benannten 'Petrinetze'[26] erwähnt, sowie der 'Calculus of Communicating Systems' von Robin Milner[27]. Eine wichtige Grundlage dieser vor allem im Rahmen von Betriebssystemforschungen entwickelten Methoden bildet der Begriff des Prozesses. Ein Prozess ist gewissermaßen das Aktivitätszentrum, das sich durch die Instruktionen eines Programms bewegt, wenn dieses sequentiell auf einem Rechner ausgeführt wird. Dieser abstrakte Prozessbegriff dient als Basis für Untersuchungen von Verhaltensweisen, wenn gleichzeitig eine Vielzahl von Prozessen in einer gemeinsamen Umgebung aktiv sind und implizit (durch Veränderung der Umgebung) oder explizit (durch

25 Insbesondere die Lithographie *Zeichnende Hände* (1948).

26 Carl Adam Petri, Kommunikation mit Automaten, Dissertation, Darmstadt 1962.

27 Vgl. Robin Milner, Communication and Concurrency, Prentice Hall, London 1989.

Senden und Empfangen von Nachrichten) miteinander kommunizieren. Da Prozesse sich in einer gemeinsamen Umgebung befinden und gemeinsame Ressourcen wie Dateien, Drucker, Prozessoren, etc. nutzen und auch verändern, wurde in diesem Bereich zwangsweise sehr früh über Modellierungsverfahren für die Organisation interagierender Prozesssysteme nachgedacht – um zum Beispiel Systemverklemmungen zu vermeiden.

Wir wollen kurz das auf E.W. Dijkstra zurückgehende und inzwischen als Klassiker der Synchronisationsproblematik geführte Beispiel der essenden Philosophen betrachten. Hierbei sitzen n ($n \geq 5$) Philosophen an einem runden Tisch. Das Leben dieser Philosophen besteht nur aus Denken und Essen. Wenn ein Philosoph hungrig wird, versucht er nacheinander seine linke und seine rechte Gabel aufzunehmen. Hat er erfolgreich beide Gabeln aufgenommen, wird er eine Weile essen, dann seine Gabeln wieder ablegen und das Denken fortsetzen. Da für n Philosophen nur n Gabeln zur Verfügung stehen, wird es Schwierigkeiten geben, sobald zwei benachbarte Philosophen gleichzeitig essen wollen. Es gilt also eine Lösung zu finden, bei der kein so genannter 'Deadlock' entsteht. Nimmt zum Beispiel zufällig jeder Philosoph nacheinander seine linke Gabel auf und gibt diese nicht wieder ab, dann kann keiner eine rechte Gabel aufnehmen und wir haben einen solchen Deadlock-Situation erreicht. Eine mögliche Lösung wäre, eine zufällige Wartezeit zu wählen und dann die Philosophen die linke Gabel wieder hinlegen zu lassen, wenn die rechte nicht frei ist. In den meisten Fällen wird das zur Lösung führen. Dennoch ist dies eine unsichere Lösung, die bei kritischen Anwendungen wie Flugsystemen oder Atomkraftwerken nicht empfehlenswert ist. Der Leser möge sich eine eigene Lösung des Problems überlegen.

Der Begriff der Heterarchie hat allerdings einen anderen Schwerpunkt, als die Begriffe und Werkzeuge, die im Zusammenhang mit der Synchronisationsproblematik und den Komplexitätsuntersuchungen (Performanz im Hinblick auf Zeit- und Ressourcenverbrauch) nebenläufiger Prozesse entwickelt wurden. Insbesondere ist der Begriff der Heterarchie in der formalen Logik verankert und hängt sehr stark mit den oben besprochenen Begriffen der Code-Reflexivität (Selbstreferenz), den Metasystemen und der Interface-Thematik zusammen. In heterarchischen Strukturen greifen die Prozesse nicht nur auf gemeinsame Daten und Ressourcen zu, sondern sie greifen in den Code der anderen Prozesse ein. Während die oben vorgestellten Metasysteme lediglich hierarchisch strukturiert in die Codes anderer Programme eingegriffen

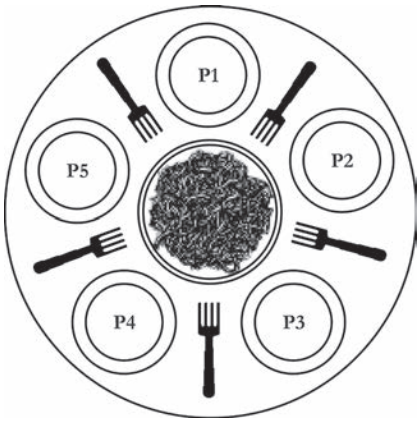


Abbildung 37:
Die essenden Philosophen von E.W. Dijkstra.

haben – die oberste Ebene bildete noch den unantastbaren Kern der Logik –, erlauben zyklische Heterarchien allgemeine Konstellationen zwischen Operanden und Operatoren. Es geht also weniger um die verklemmungsfreie Distribution von Prozessen, als vielmehr um die Distribution von miteinander verbundenen Logiken. Theoretische Überlegungen hierzu wurden aber bisher weniger von den Computerwissenschaften vorgelegt, als vielmehr von der Systemtheorie, der Kybernetik und der theoretischen Biologie.

Von interaktiven Anwendungen im Feld der neuen Medien wird ständig größere Entwicklungsfähigkeit und Offenheit in der Interaktion erwartet. Hierbei stellt sich die Frage, wie wir solche Systeme konstruieren können, die sich einerseits kontrollieren lassen, also zuverlässig, robust und konsistent sind, und die andererseits interessantes und überraschendes Verhalten generieren. Heterarchien sind in dieser Hinsicht ein interessanter Ansatz, da sie sich nicht mit der bloßen Abarbeitung von Rechenaufgaben beschäftigen, sondern eine Dimension des 'Werdens' beinhalten. Im Zentrum heterarchischer Betrachtungen geht es um Fragen von Kontrolle vs. Autonomie, um die Standpunktabhängigkeit von Wahrheitsaussagen, die Emergenz von Phänomenen und die maschinelle Erzeugung 'echter' Alternativen – kurz, es geht um eine Dimension des Werdens auf der Basis von formalen Systemen. Dies soll auch das Bild *Die Hand unterwirft sich dem Intellekt* von Carlo Maria Mariani deutlich machen. Wir müssen Organisationsformen entwickeln, die so miteinander wechselwirken, dass neue Qualitäten entstehen. Die Arbeiten zu diesen Themen stehen in der Tradition der *Second Order Cybernetics* und hier insbesondere des Philosophen Gotthard Günther.[28] Rudolf Kaehr spricht im Zusammenhang mit Logik-Forschungen von der Domestikation von Antinomien. Im Hinblick auf Heterarchien stellt er fest:[29] „Was uns bis dahin als Selbstreferentialität erschien und Kopfschmerzen produzierte, das ist im Grunde genommen etwas Sekundäres. Primär ist nicht die Zirkularität und ihre Antinomien, sondern die Verteilung von Logiken."

Einiges, das im Rahmen verteilter Logiken diskutiert wird, ist heute bereits im Bereich verteilter und vernetzter Anwendungen gängige Praxis, ohne allerdings dort mit den entsprechenden Begriffen belegt zu sein. Wir hatten zum Beispiel gesehen, dass aus dem chiastischen Spiel zwischen Zeichen als Operanden und Zeichen als Operatoren im Nacheinander der Verarbeitung die eigentliche Mächtigkeit der programmierbaren Maschine erwächst. Eine operationsfähige Dialektik, wie Gotthard Günther sie fordert, wäre allerdings in der Lage, nicht nur das 'Nacheinander' der Ordnungsrelationen zu realisieren, sondern auch das 'Zugleich'. In welchen Fällen dieses Zugleich auf sequentielle Prozesse reduziert werden kann und in welchen es tatsächlich irreduzibel bleibt, d.h. nicht mehr

28 Gotthard Günther, Beiträge zur Grundlegung einer operationsfähigen Dialektik, Band 1–3, Felix Meiner Verlag, Hamburg 1976 – 1980.

29 In: Kalküle für Selbstreferentialität oder selbstreferentielle Kalküle, in: Reader zur Ringvorlesung Radikaler Konstruktivismus, S. 33.

simuliert werden kann, muss noch untersucht werden. Die minimale nicht-transitive Heterarchie McCullochs ist jedenfalls heute – implizit oder explizit – in fast jedem Computerspiel zu finden und ohne weiteres, wie McCulloch dies selbst in seinem Brief an den Herausgeber darlegt, mit drei logischen UND-Gattern realisierbar, aber eben nicht ohne übergreifende Diallele.

Die im folgenden Abschnitt behandelten 'Interfaces' bedeuten eine wesentliche Erweiterung der in diesem Abschnitt dargestellten Struktur von Rechensystemen. Mit der Thematisierung des Interfaces rückt die Frage ins Zentrum, wie wir überhaupt zu den Zeichen kommen, d.h. welche Mechanismen zwischen den Zeichen, die in den Rechensystemen verarbeitet werden, und der offenen und komplexen Außenwelt vermitteln. Interfaces sind Abbildungsverfahren, die auf der Seite der Eingabe (des Inputs) die 'Verkürzung' und zeichenhafte Darstellung der Außenwelt realisieren. Aus der Vielfalt des Umweltgeschehens werden einzelne Aspekte isoliert und als Zeichen codiert. Ausgabeseitig haben wir es mit einer Umkehrung dieses Prozesses zu tun. Numerische Werte und Zeichen werden in nicht-zeichenhafte Wahrnehmungen transformiert. Natürlich können die solchermaßen wahrnehmbaren Oberflächen des Systems wiederum semiotisch interpretiert werden. Zwischen den Zeichen, mit denen das Systems rechnet und den semiotisch interpretierten Zeichen des Betrachters liegt die Transformation durch das Interface. Diese Abbildung wollen wir im nachfolgenden als 'Interface-Mapping' bezeichnen.

1.3.2 Interfaces[30]

„Ein Artefakt kann als Punkt der Begegnung – in der heutigen Terminologie: 'als Schnittstelle' – zwischen einer 'inneren' Umgebung, der Substanz und inneren Gliederung des Artefakts selbst, und einer 'äußeren' Umgebung, der Umwelt in der es operiert, gedacht werden."

Herbert A. Simon[31]

Zwei Aspekte der Schnittstelle sollen im Folgenden herausgearbeitet werden: 1. Die Prinzipien der 'Interaktivität' und der 'Interaktion' und ihre Bedeutung für die Theorie der klassischen Universalmaschine. 2. Der Begriff des 'Interface-Mappings'. Das sind die Abbildungsmechanismen, die in den Mensch-Computer-Schnittstellen aktiv sind und die Ereignisse aus der Umgebung der Maschine in Zeichenfolgen umwandeln und umgekehrt. Ganz im Sinne Herbert Simons gehen wir dabei davon aus, dass in heutigen technischen Systemen der Begriff der Schnittstelle nur im Hinblick auf den Zweck des Systems betrachtet werden kann. Nicht nur Computer, sondern jedes Artefakt zeigt erst an der Schnittstelle, ob seine innere Struktur und die Umgebung in angemessener Weise

30 Die Begriffe 'Schnittstelle' und 'Interface' werden hier als synonym gesehen.

31 In: Die Wissenschaften vom Künstlichen, S. 8.

zusammenwirken, nämlich so, dass sich das bezweckte Verhalten einstellt. „Wenn die innere Umgebung der äußeren angemessen ist oder umgekehrt, dann wird das Artefakt seinen Bestimmungszweck erfüllen. Ist die Uhr unempfindlich gegen Stöße, so ist sie als Schiffschronometer brauchbar (und umgekehrt: ist sie es nicht, können wir sie durch Aufstellen auf dem heimischen Kaminsims retten).“[32] Die Frage der Angemessenheit eines Interfaces wird also nicht in erster Linie durch das Artefakt und seine Umwelt bestimmt, sondern durch den Zweck, den der Konstrukteur oder Nutzer vorgibt. Im Rahmen dieser Zweckbestimmtheit wird das Innere eines Artefakts die Komplexität seiner Umgebung widerspiegeln. Im weiteren Verlauf des Abschnitts werden wir aber mit den Arbeiten von Peter Cariani einen avancierten Schnittstellenansatz kennen lernen, bei dem der vorgegebene Zweck des Systems nicht mehr im Zentrum steht.

1.3.2.1 Interaktivität & Interaktion

> *“Algorithms and Turing machines (TMs) have been the dominant model of computation during the first 50 years of computer science, playing a central role in establishing the discipline and providing a deep foundation for theoretical computer science. We claim that TMs are too weak to express interaction of object-oriented and distributed systems, and propose interaction machines (IMs) as a stronger model that better captures computational behavior for finite interactive computing agents.”*
>
> Peter Wegner, Dina Goldin[33]

Wir wollen zunächst *Interaktivität* von *Interaktion* unterscheiden. Interaktion ist ursprünglich ein Begriff aus der Soziologie, der mit Namen wie George Caspar Homans, Talcott Parsons und George Herbert Mead verbunden ist und das aufeinander bezogene Handeln von zwei oder mehreren Personen beschreibt. Interaktivität bezieht sich lediglich auf die technische Fähigkeit eines Programms, während seiner Laufzeit neue Eingaben zu verarbeiten. Der Begriff der Interaktion soll dagegen für Kommunikationsaspekte reserviert bleiben. Auf die Mensch-Maschine-Kommunikation verallgemeinert könnte man sagen, dass mit dem Begriff der Interaktion Verhalten beschrieben wird, das eine Menge von symbolischen Nachrichten umfasst, die innerhalb einer Menge von Objekten (Menschen wie Maschinen) in einem bestimmten Kontext ausgetauscht werden, um ein bestimmtes Ziel zu erreichen. Interaktion fokussiert damit auf die Bedeutung und die Intentionalität eines Programms und bezeichnet deshalb in erster Linie eine kognitive Handlung mit symbolischen Inhalten. Technische Interaktivität zusammen mit leistungsfähigen graphischen und akustischen Ein-/Ausgabe-Apparaturen sind zwar keine hinreichenden, aber doch notwendige Voraussetzungen für Interaktion. Interaktive Anwendungen zielen immer schon auf symbiotischen Nutzen durch Interaktion ab, d.h. das hybride System aus Mensch

32 Ebd.

33 Peter Wegner, Dina Goldin, Interaction, Computability, and Church’s Thesis.

und Maschine soll als Gesamtsystem in der Lage sein, Aufgaben zu bewältigen, die der Mensch alleine nicht in Angriff nehmen könnte. Diese Vision stand auch bereits in den 1960-er Jahren im Zentrum der Engelbartschen Arbeiten, die er unter dem Titel „Augmenting Human Intellect" durchführte und die neben vielen anderen technischen Neuerungen zur Entwicklung der Computermaus führten.[34]

Peter Wegner wirft die Frage auf, wie Interaktivität, die im Umgang mit Computern längst alltägliche Praxis ist, sich mit dem klassischen Begriff der Berechenbarkeit verträgt. Seine erweiterte Definition von Berechenbarkeit, die er aus dieser Fragestellung entwickelt, unterscheidet sich deutlich von klassischen Begriffen, wie sie von Turing, Church, Markov, u.a. eingeführt wurden. Die von Wegner eingeführten Interaktionsmaschinen sind Turingmaschinen, die um Ein-/Ausgabe-Ströme erweitert werden. In mancher Hinsicht sind Interaktionsmaschinen vergleichbar mit den älteren, bekannten Orakelmaschinen, die ebenfalls Entscheidungen außerhalb der Maschine in die Berechnungen mit einbeziehen. Dieser Sichtwechsel vom klassischen Begriff des Algorithmus hin zur Interaktion vollzieht den technologischen Wechsel nach, der in der Praxis in den zurückliegenden Jahrzehnten von Mainframes und Number-Crunchern hin zu Workstations, Netzwerken und eingebetteten Systemen mit graphischen Benutzerschnittstellen stattgefunden hat. „The evolution of computer technology from the 1970s to the 1990s is captured by a paradigm shift from algorithms to interaction. Algorithms yield outputs completely determined by their inputs that are memoryless and history independent, while interactive systems like personal computers, airline reservation systems, and robots provide history-dependent services over time that can learn from and adapt to experience."[35] Das klassische Algorithmenmodell bildet einen Softwarebegriff ab, der in der Praxis unter dem Begriff der 'Batchsoftware' bekannt ist und bis in die 1970-er Jahre dem Standard entsprach. Bei Batchsoftware gibt es während der gesamten Programmausführung keine Möglichkeit, in die Berechnungen einzugreifen. Die Software wird zusammen mit den zu bearbeitenden Daten als kompletter Auftrag an die Maschine übergeben, die ihre Berechnungen selbständig bis zum Erreichen des Abbruchkriteriums durchführt. 'Interaktive Software' bietet Möglichkeiten an, die Programmausführung durch Benutzereingaben während des laufenden Betriebs zu beeinflussen. Wichtig an Wegners neuem Maschinenmodell ist, dass es – auch wenn es den Rahmen der klassischen Algorithmentheorie nicht sprengt – die derzeitige Praxis der Maschine sehr viel besser abbildet als die klassischen Modelle, die den Begriff der Interaktivität nicht kennen. Interaktionsmaschinen bieten eine praktikable Möglichkeit, die Interaktion in objektorientierten und verteilten Systemen angemessen zu beschreiben.

34 Siehe Zwischenspiel: Interfaces – Von Menschen und Mäusen.

35 Peter Wegner, The Paradigm Shift from Algorithms to Interaction.

1.3.2.2 Interface-Mapping

„Any machine is prisoner of its input and output domain."
Allen Newell[36]

Heinz von Foerster beschreibt das fundamentale Prinzip der 'undifferenzierten Codierung' – das bereits Mitte des 19. Jahrhunderts vom deutschen Neurophysiologen Johannes Müller formuliert wurde und heute die wichtigste Basis für die gesamte Theorie des Radikalen Konstruktivismus bildet – folgendermaßen:[37] „Die Erregungszustände einer Nervenzelle codieren nur die Intensität, aber nicht die Natur der Erregungsursache. Codiert wird nur: 'So-und-so viel an dieser Stelle meines Körpers' aber nicht 'was'."[38] Durch Messungen weiß man heute sehr genau, dass neuronale Erregungen die Form von elektrochemischen Spannungsverläufen mit spitzen Impulsen haben. Dabei ist die Anzahl der Impulse in einer bestimmten Zeitspanne abhängig von der Reizstärke. So führt zum Beispiel ein schwacher Lichteinfall in der Retina zu relativ seltenen Impulsen, ein starker Lichteinfall dagegen zu häufigen. Das heißt, im Nervensignal wird ausschließlich die Quantität einer Reizung durch die Frequenz des Signals codiert, nichts sonst. In diesem Signal werden damit weder Sinnesmodalität (Sehen, Hören, Schmecken, Riechen, Tasten) noch Sinnesqualität (rote oder gelbe Farbe, hoher oder tiefer Ton) unterschieden. Das so genannte topologische Prinzip bedeutet, dass das Gehirn aus dem Ort dieser Reizung und seiner Häufigkeit die Sinnesqualitäten konstruieren muss.[39] Das Prinzip der undifferenzierten Codierung in natürlichen Organismen gilt aber nicht nur eingabeseitig, sondern gleichermaßen für die Impulse, die an den motorischen Apparat gesendet werden und diesen kontrollieren, wie auch für die Aktivität des Gehirns selbst. Die Nervenzelle ist damit ein echtes Ja-Nein-Organ, das eine quantitative Größe übermittelt, indem es diese als mehr oder minder dichte Folge von Einzelimpulsen zerlegt. Die Reizstärke ist hierbei direkt proportional zur Anzahl der generierten Impulse. John von Neumann weist bereits 1951 in seiner „Allgemeinen und logischen Theorie der Automaten"[40] auf den interessanten Unterschied zwischen der Codierung in Nervenzellen und der Codierung in Computern hin. Während Nervenzellen die 'Zählmethode' bevorzugen, benutzen Computer die Methode der 'Dualbruchentwicklung'. Die Zählmethode ist zwar unwirtschaftlicher, da eine physikalische Größe mit einer Million unterscheidbaren Ereignissen auch tatsächlich eine Million Impulse benötigt. Mit der Dualzahlentwicklung kann eine Zahl dieser Größe dagegen mit nur 20 Bits dargestellt werden. Der entscheidende Vorteil der Zählmethode ist ihre hohe Fehlertoleranz gegenüber der Entwicklungsmethode. Während bei

36 In: Physical Symbol Systems, Cognitive Science 4(2), 1980, S. 135 – 183.

37 Erkenntnistheorien und Selbstorganisation, in: Siegfried Schmidt (Hrsg.), Der Diskurs des radikalen Konstruktivismus, S. 138.

38 Siehe auch Zwischenspiel: Logische Kalküle, Perzeptren und neuronale Netze.

39 Ebd.

40 The General and Logical Theory of Automata (1951), in Deutsch erschienen in: Hans Magnus Enzensberger (Hrsg.), Kursbuch 8, S. 139 – 175.

der Zählmethode der Ausfall eines einzelnen Impulses keine große Verfälschung bedeutet, kann bei einer Dezimalzahl ein einziger Fehler den übermittelten Wert um ganze Größenordnungen verschieben (wenn z.B. in der Zahl 6430000 statt der 6 eine 7 übertragen wird, verschiebt sich das Ergebnis um Größenordnung eine Million).

Das Interface des Menschen zu seiner Umwelt besteht nach der Theorie der undifferenzierten Codierung ausschließlich aus frequenzcodierten Impulsen. Damit ist ein grundsätzlicher Zusammenhang zwischen Interfaces und Selbstreferentialität gegeben. Das Gehirn hat keinen direkten Zugang zur Welt und kann deshalb nichts von außen abbilden, sondern nur etwas aus sich heraus konstruieren. Die Theorie der Selbstreferentialität des Gehirns wird unter dem Begriff der 'Autopoiese' subsumiert. Man spricht in diesem Zusammenhang von struktureller Kopplung zwischen dem System und seiner Außenwelt. Solche geschlossenen Systeme registrieren zwar die Veränderungen in der Umwelt, sind jedoch autonom bezüglich ihres eigenen Verhaltens, d.h. ob sie ihr Verhalten und ihre innere Struktur aufgrund der Störungen (Perturbationen) durch die Außenwelt ändern, bleibt ihnen selbst überlassen. Die Umgebung der programmierbaren Maschine ist ähnlich einfach wie das Interface des Menschen in der Theorie der undifferenzierten Codierung, sie besteht aus Zeichen, die numerische Werte und Texte codieren. Dennoch ist die Ausgangsituation für die beiden 'Systeme' vollkommen verschieden. Aufgrund ihrer vollständig auf Zeichen basierenden Natur kann die klassische Universalmaschine nur Probleme bearbeiten, die bereits als Zeichen codiert sind. Im Gegensatz zum Gehirn erzeugt die Maschine bisher keine neuen semantischen Kategorien, sondern bewegt sich in einer durch den Programmierer objektivierten Welt. Alles, was die klassische Maschine damit tut, ist Zeichen nach vorgegebenen Regeln zu manipulieren. Die Zeichen stehen für bereits objektivierte Sachverhalte der Außenwelt. Diese Tatsache dient Kritikern der Maschine häufig als Argument, dass die Maschine nur stur ausführen kann, was ihr vorher befohlen wurde. Maschinen könnten zwar Aufgaben erledigen, die sich in ein starres Schema gießen lassen und damit sogar bis zu einem gewissen Grad Vernunft simulieren, sie könnten aber nie etwas hervorbringen, das als originell oder kreativ zu bezeichnen wäre. Aufgrund obiger Ausführungen zu Metasystemen sollte allerdings klar sein, dass die Verhältnisse nicht ganz so einfach sind. Bereits in den 1960-er Jahren wehrt sich der große ungarische Kybernetiker und Ingenieur Tihamér Nemes gegen die von Martin Gardner vorgebrachte Kritik, dass bisher außer in wissenschaftlichen Zukunftsromanen noch keine einzige logische Maschine und kein Rechenautomat auch nur den geringsten Funken Originalität gezeigt hat. Auch das bekannte Argument von Ada A. Lovelace – landläufig auch als Erfinderin der Programmierung bekannt – über die *Analytische Maschine* von Charles Babbage wird zitiert: [41] „Die Maschine beansprucht nicht, irgendetwas

41 Siehe: Kybernetische Maschinen, S. 237 f.

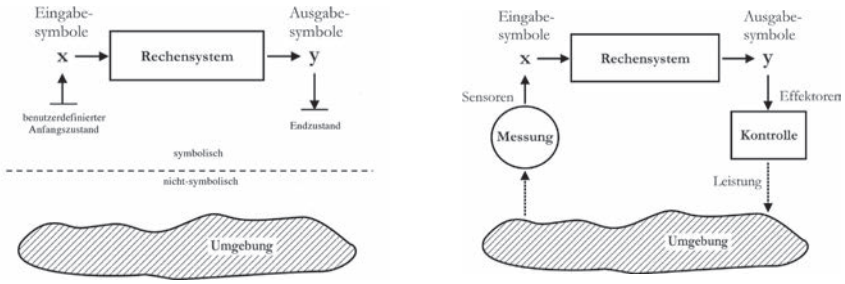


Abbildung 38/39:
Klassische symbolverarbeitende Maschine, es werden nur Daten akzeptiert, die bereits in symbolischer Form vorliegen (links). Symbolverarbeitende Maschine mit vorgeschalteten Sensoren und nachgeschalteten Effektoren (rechts).

schaffen zu wollen. Sie kann nur ausführen, was ihr befohlen werden kann. Sie kann Analyse simulieren, besitzt aber nicht die Fähigkeit, eine analytische Relation oder Wahrheit zu finden."

Nemes lehnt die Vorstellung von Originalität als metaphysisches Etwas ab und geht davon aus, dass sie ihre eigenen Naturgesetze hat, die erforscht und nachgebildet werden können. Er fragt in Konsequenz: „Was wäre aber, wenn man den Mechanismus der Originalität selbst in die Maschine einbauen würde?" Dies ist nichts anderes als die Forderung nach einem Metasystem. Nicht mehr die Regeln zur Lösung eines einzelnen Problems werden implementiert, sondern dieses Problem wird eingebettet in ein größeres System, das die Lösung des konkreten Problems steuert. Der Rahmen des formalen Systems wird also nicht gesprengt, sondern nur vergrößert.

Welche Funktionen erfüllen Interfaces in diesem Zusammenhang? Ihre übliche Aufgabe wird darin bestehen, dem Benutzer die Last abzunehmen, nur vermittels Zeichen mit der Maschine zu interagieren, sondern diese Zeichen eingabeseitig mittels Sensoren automatisch aus der Umwelt zu extrahieren und ausgabeseitig die Zeichen mittels Effektoren in Handlungen oder andere Formen der Darstellung, z.B. mit bilderzeugenden Apparaten umzusetzen. Die inneren symbolischen Strukturen müssen dabei in irgendeiner Weise die Schnittstelle modellieren. So verlangen zum Beispiel Texte eine andere interne Repräsentation und andere Verarbeitungsmechanismen als Bilder. Im Bezug auf das symbolische und nicht-symbolische Ein-/Ausgabe-Verhalten von Maschinen lassen sich damit vier Spielarten unterscheiden. Die klassische symbolverarbeitende Universalmaschine transformiert symbolische Eingaben in symbolische Ausgaben. Sensoren transformieren nicht-symbolische Ereignisse der Außenwelt in symbolische Darstellungen und Effektoren transformieren symbolische Ereignisse in nicht-symbolische Handlungen oder Darstellungen. Vollständig nicht-symbolische Interaktionen sind mit digitalen Computern nicht realisierbar.

Für die Zwecke der digitalen Verarbeitung genügt es aus Sicht der Interfaces, die Umwelt auf die Gesamtheit aller physikalisch messbaren Größen zu reduzieren.

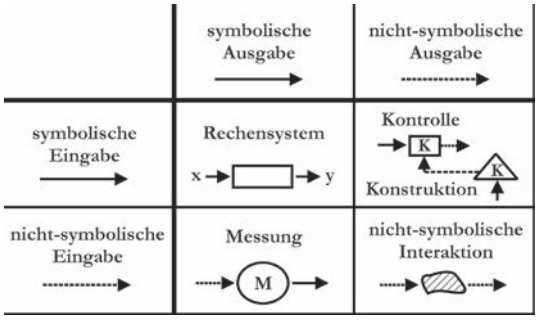


Abbildung 40:
Symbolische und nicht-symbolische Verarbeitungsvarianten.[42]

Ein Objekt oder Ereignis im Computer wird so durch eine Menge von Funktionen beschrieben. In der Hard- und Software des Interfaces findet die Reduktion der Umwelt auf die gewählten Funktionen statt. Durch diesen Vorgang wird die Umwelt in Objekte mit bestimmten Eigenschaften transformiert, sie wird auf diese Weise 'objektiviert'. Indem wir etwas derart ins Auge fassen, müssen wir notwendigerweise alle anderen Möglichkeiten außer acht lassen. Der Abbildungsmechanismus im Interface erzeugt – mit anderen Worten – eine permanente Blindheit gegenüber den nicht zur Messung ausgewählten Phänomenen. Es ist nicht möglich, Programme zu schreiben, die dieser Voraussetzung entgehen können. Oder, umgekehrt gesagt, ein Programm schreibt die 'Blindheit' fest, solange es nicht 'lernfähig' ist. Was nicht gemessen wird, existiert somit auch nicht und steht nicht für die direkte Verarbeitung zur Verfügung. Im Gegensatz zum menschlichen Gehirn sind Computerprogramme jedoch operational offene Systeme, d.h. das Ein-/Ausgabe-Verhalten ist extern determiniert. Der Programmierer legt die Grenze zwischen System und Umgebung fest und bestimmt durch den inneren Aufbau des Interfaces, welche Ereignisse in Form von Symbolen die Maschine erreichen oder sie verlassen. Sowohl das Zeichenalphabet, als auch die Bedeutung der Zeichen, die diese fiktive Grenze passieren, sind in heutigen technischen Systemen üblicherweise fest vorgegeben. Wenn aber die Messfunktion des Interfaces fest verdrahtet ist, existiert die Umgebung nur vom Standpunkt des Systemkonstrukteurs aus und nicht vom Standpunkt des Systems selbst. Das System kann zum Beispiel weder Zeichen, die permanent durch Interface-Funktion generiert werden, verweigern, noch wird es Zeichen umdefinieren oder neue Zeichen erfinden. In dieser Hinsicht ist jedes System Gefangener seines eigenen Interfaces. Wer aber nur einen Hammer zur Verfügung hat – so sagt der Volksmund – neigt dazu, alles wie einen Nagel zu behandeln. Von einer 'echten' Umgebung müssen wir allerdings fordern, dass Unvorhergesehenes geschehen kann, das nicht schon im Interface-Entwurf berücksichtigt ist.

Aber auch diese starren Schnittstellen werden für den Benutzer, im Vergleich zur vorher gebräuchlichen zeichenbasierten Interaktion, bereits enorm wertvoll sein. Ihre Bedeutung liegt darin, dass dem Benutzer nicht-symbolische Interaktionen mit der Maschine ermöglicht werden – wenn auch in einem Rahmen, der durch

42 Vgl. Peter Cariani, On the Design of Devices with Emergent Semantic Functions.

den Entwurf des Systems vorgegeben ist. Der Umgang mit der Maschine verlagert sich so vom zeichenbasierten Ein-/Ausgabe-Verhalten hin zu visuellen, akustischen und taktilen Wahrnehmungsprozessen und damit zu soziologisch und kulturell geprägter Handlung und Kommunikation. Daraus folgt aber auch, dass die in den Interfaces wirksamen Funktionen einen eigenen neuen Forschungsgegenstand mit eigenen Fragen und Methoden darstellen. Dem Interface-Mapping, also der Abbildung der Außenwelt in maschinell verarbeitbare Symbole (und umgekehrt), muss dabei noch weitaus mehr Aufmerksamkeit geschenkt werden als bisher. Wenn die Interaktion zwischen Mensch und Maschine gelingen soll, dann müssen weit jenseits bisher fokussierter ergonomischer Belange Wahrnehmung und Kognition im Interface eine neue Einheit bilden.

1.3.2.3 Zur Zukunft des Interfaces

„To build devices, which find new observational primitives for us, they must be made epistemically autonomous relative to us, capable of searching realms for which we have no inkling."
Peter Cariani[43]

Die Bedeutung des Interfaces liegt in seiner Eigenschaft, eine Verbindung zwischen symbolischen mit nicht-symbolischen Welten herzustellen. Mit Hilfe des in den Schnittstellen aktiven Interface-Mappings können Aufgaben von der Maschine bearbeitet werden, die nicht schon in symbolischer Form vorliegen. Wir hatten gesehen, dass die klassische Universalmaschine im Hinblick auf ihre Problemlösungskompetenz ein relativ starres System ist. Die Turingmaschine oder auch die Interaktionsmaschine sind zwar aufgrund ihrer Programmierbarkeit fähig, jede rekursive Funktion zu berechnen, aber sie werden dabei nicht ihre eigene innere Struktur verändern, um zum Beispiel ihre eigene Performanz zu verbessern. Durch die Verbindung der internen Symbolverarbeitung mit der Außenwelt mittels Interfaces erweitert sich der Anwendungshorizont der Maschinen dennoch drastisch. Mit Hilfe der Schnittstellen werden Echtzeitanwendungen möglich, die Ereignisse im Moment ihres Auftretens aus der Umgebung extrahieren und der symbolischen Verarbeitung zuführen. Mittels Effektoren können dann die Berechnungsergebnisse wieder als nicht-symbolische Handlungen in die Umgebung zurückwirken. Gelingt es, diesen Elementarzyklus in ausreichender Geschwindigkeit zu durchlaufen, kann die Maschine in vollkommen neuen Handlungszusammenhängen Aufgaben erfüllen. Der symbolische Verarbeitungsstrom ist hier eingebettet in die Umwelt und wirkt vermittels der von ihm gesteuerten Handlungen in der Umgebung indirekt auf sich selbst zurück. Trotzdem läuft hierbei die interne Verarbeitung noch nach starren Regeln

43 Ebd.

ab, die sich erst ändern, wenn ein neues Programm geladen wird. Auch die vor- und nachgeschalteten Interfaces ändern nichts an dieser Tatsache. Eine grundlegende Änderung der Situation erhalten wir erst, wenn wir die oben diskutierten Codereflexivitäten und Metasysteme ins Spiel bringen.

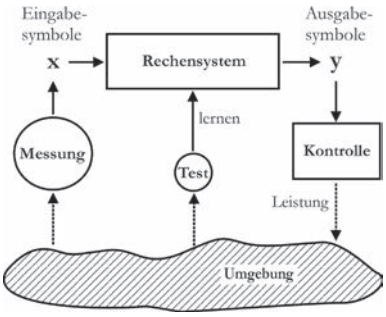


Abbildung 41:
Code-Adaptivität. Der interne Algorithmus wird aufgrund von Leistungsmessungen so verändert, dass eine verbesserte Performanz des Systems in seiner Umwelt resultiert.

Sobald das System in die Lage versetzt ist, seine eigene Performanz zu messen, können die bekannten Strategien zum Einsatz kommen, um die symbolisch repräsentierten Verarbeitungsroutinen des Programms zu modifizieren. Aufgrund der Beobachtung der Leistung des Systems in seiner Umwelt können reflexive Systeme und Metasysteme gezielt die symbolischen Repräsentationen der Programme verändern und diese dann erneut auf ihre veränderte Leistung hin messen. Auf diesem Prinzip beruhen alle algorithmischen Evolutionsstrategien und genetische Algorithmen. In den meisten Anwendungen solcher evolutionärer Strategien wird die Umgebung des Systems jedoch nicht eine reale Umwelt sein, sondern eine simulierte, wobei die Performanz des Systems sehr einfach mittels einer so genannten Fitness-Funktion gemessen werden kann.[44] Adaptive Systeme besitzen die Flexibilität, sich an neue, bis zu einem bestimmten Grad unvorhergesehene Situationen anzupassen. Zumindest wird der Programmierer davon befreit, die Ereignisse der Umwelt bis ins Detail vorhersehen zu müssen. Die Adaptionsfähigkeit des Systems wird allerdings stark davon abhängen, welche Strategien der Code-Reflexion angewendet werden.

An dieser Stelle ist eine metalogische Bemerkung im Sinne Batesons angebracht. Der Aufbau unserer bisherigen Argumentationslinie von der klassischen Turing-Verarbeitung über Interface-Systeme hin zu adaptiven Systemen sollte hauptsächlich dazu dienen, zu einer von hieraus einfachen Erkenntnis zu führen, nämlich, dass in den bisherigen Adaptionsstrategien zur Erreichung flexiblen Verhaltens ein wesentlicher Bereich vernachlässigt wurde – das Interface. Es gibt keinen Grund, warum lediglich symbolische Repräsentationen der Algorithmen einer Reflexion unterzogen werden, und nicht auch die Mess-, Kontroll- und Darstellungsfunktionen in den Interfaces. Im Extremfall könnte dies soweit gehen, dass das System Konstruktionseinheiten enthält, um neue Interfaces physikalisch zu bauen und an die Maschine anzuschließen. Die Idee der inzwischen schon kommerziell verfügbaren 3D-Drucker, die anhand von Konstruktionsplänen physikalische Realisierungen von Objekten 'ausdrucken', zeigt bereits in diese Richtung. Es fehlt lediglich noch die vollständige Schließung des Kreises, d.h.

44 Siehe Zwischenspiel: Digitaler Darwinismus.

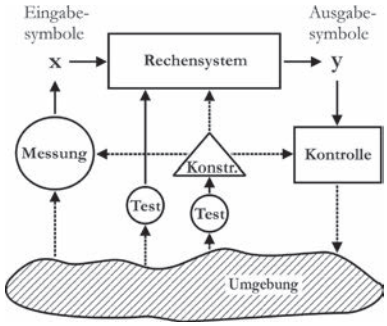


Abbildung 42:
Interface-Adaptivität. Nicht nur der symbolische Bereich der Anwendung unterliegt der Modifikationsstrategie, sondern auch das Interface-Mapping.

eine Maschine, die durch eigene Beobachtung ihrer Performanz neue Interfaces entwirft, diese herstellt und sich selbst daran anschließt.

1.3.3 Flottierende Codes

„Zur Zeit kann ein Mensch nur durch materielle Übermittlung und noch nicht als Nachricht von einem Ort zum anderen gelangen. Indessen dient sogar heute schon die Übermittlung von Nachrichten dazu, eine Erweiterung seiner Wahrnehmungs- und Aktionsmöglichkeiten über die ganze Welt zu ermöglichen. Ist diese Unterscheidung zwischen materieller Übermittlung und Nachrichtenübermittlung unbedingt ewig und unüberbrückbar?"
NorbertWiener[45]

In der Computerpraxis vor 1970 stand die teure Hardware im Zentrum der Aufmerksamkeit und Software wurde als kostenlose Zugabe betrachtet. Erst ab ca. 1970 kann man ernsthaft von Softwareentwicklung als Disziplin und einer eigenständigen Softwarebranche sprechen. Aber auch in der klassischen Berechenbarkeitstheorie, etwa der Turingmaschine, stand die Maschine noch im Mittelpunkt und das Programm war an den Gegebenheiten der Hardware ausgerichtet. Der Ort des Geschehens war die singuläre Hardware, die in eine Umgebung eingebettet und mit einer Schnittstelle ausgestattet war, über die sie unter anderem ihr Programm einlas und die Berechnungsdaten ausgab. Ganz anders die Situation heute. Computer sind vernetzt und omnipräsent. Die Informatik spricht inzwischen vom neuen Leitbild des 'Pervasive Computing', d.h. allgegenwärtigen, miniaturisierten, vernetzten und umgebungssensitiven Mikrochips in der Kleidung, in Accessoires und Haushaltsgegenständen, nicht zuletzt im menschlichen Körper. Konzeptionell bringt dies eine Befreiung der Software mit sich und rückt sie gleichzeitig in den Fokus. Die Hardware wird Teil der Umgebung und eine mobile Software nimmt die zentrale Position ein.

45 In: Mensch und Menschmaschine, S. 96.

Die Folge sind 'flottierende Codes' als wesentliches Zeichen der neuen hybriden Maschine, d.h. Programme, die nicht mehr an einen Ort gebunden sind, sondern sich frei durch die Netze bewegen und Hardware als wählbare Umgebung betrachten. Natürlich müssen auf der Seite der Hardware eine Vielzahl von Bedingungen erfüllt werden, damit flottierende Codes realisiert werden können. Im Zentrum der Betrachtungen steht nun aber nicht mehr die Hardware, sondern die Software und die Gesamtheit der vernetzten Maschinen bilden zusammen mit ihren Interfaces (Sensoren und Effektoren) die Ressourcen für diese Codes. Wesentliche Kompetenzen, die solche Programme – so genannte Softwareagenten – mitbringen müssen, sind erstens die Fähigkeit, sich durch die Netze zu bewegen und zweitens, mit anderen Agenten zu kommunizieren. „An agent is a computational entity such as a software program or a robot that can be viewed as perceiving and acting upon its environment and that is autonomous in that its behavior at least partially depends on its own experience. As an intelligent entity, an agent operates flexibly and rationally in a variety of environmental circumstances given its perceptual and effectual equipment."[46] „Agents operate and exist in some environment, which typically is both computational and physical. The environment might be open or closed, and it might or might not contain other agents. Although there are situations where an agent can operate usefully by itself, the increasing interconnection and networking of computers is making such situations rare, and in the usual state of affairs the agent interacts with other agents."[47] Damit mobile Agenten von einem Punkt des Netzes zu einem anderen wandern können, müssen die beteiligten Rechner in der Lage sein, Agenten zu verpacken und an andere Rechner zu versenden. Des weiteren müssen sie Agenten empfangen, überprüfen und ausführen können. Die Umgebung solcher Multiagentensysteme besitzt eine Reihe typischer Merkmale. Üblicherweise sind diese Umgebungen offen und es existiert keine zentrale Kontrolle oder ein zentraler Entwurf des Gesamtsystems. Es ist nicht bekannt, welche oder wie viele Agenten sich zu einem bestimmten Zeitpunkt im Netz befinden und welche Interessen sie verfolgen. Die Umgebung bietet in der Regel heterogene Kommunikationsdienste und Interaktionsprotokolle an. Diese Gegebenheiten stellen große Anforderungen an den Entwurf von Agenten. Erst durch diese Mobilität können Agenten ihr wirkliches Potenzial entfalten. Um ihre Aufgaben zu erfüllen, müssen Agenten nicht mehr eine Vielzahl von Anfragen, Antworten und Aufträgen durch das Netz verschicken und empfangen, sondern sie begeben sich direkt vor Ort, um ihre Aufgaben zu erledigen. Dies bewirkt nur eine einmalige Netzbelastung und die Netzverbindung muss nicht über die gesamte Zeit aufrecht erhalten werden.

Multiagentensysteme bilden nur die anspruchsvollste wissenschaftliche Front flottierender Codes. Es gibt eine Vielzahl einfacherer alltäglicher Anwendungen, die ebenfalls diese neue Flexibilität des Codes zeigen. Zum Beispiel macht sich

46 Gerhard Weiss (Hrsg.), Multiagent Systems, S. 1.

47 Ebd., Michael N. Huhns, Larry M. Stephens, Multiagent Systems and Societies of Agents, S. 79.

bereits jeder Web Browser dieses Prinzip zunutze, wenn er ein Java-Applet über das Netz lädt und auf der Maschine vor Ort zur Ausführung bringt. Auch Viren und Würmer sind bekannte Beispiele flottierender Codes. Während Würmer noch auf die Verbreitung durch die Benutzer angewiesen sind und lediglich darauf aus sind, deren Zeit zu stehlen, sind Viren weitaus aggressiver. Sie sind in der Lage, fremden Code zu infizieren, d.h. sich in diesen einzuschreiben und ihn zu versklaven. Flottierende Codes sind aber, wie wir bereits gesehen haben, nur ein Merkmal der hybriden Maschine. Für die erfolgreiche Realisierung von Multiagentensystemen sind eine Reihe anderer wirksamer Mechanismen notwendig, die wir weiter oben schon kennen gelernt haben. So können etwa Agenten, die sich autonom verhalten, frei durch die Netze bewegen, mit anderen Agenten kooperieren und daraus eigenständig Schlussfolgerungen ziehen, nur realisiert werden, wenn gleichzeitig selbstreferentielle bzw. heterarchische Prinzipien und/oder Metasysteme zum Einsatz kommen.

1.4 Konsequenzen für die künstlerische und kulturelle Praxis der neuen Medien

„To understand the logic of new media, we need to turn to computer science.
It is there that we may expect to find the new terms, categories,
and operations that characterize media that became programmable.
From media studies, we move to something that can be called 'software studies' –
from media theory to software theory."
Lev Manovich[1]

Nachdem wir im letzten Abschnitt wichtige Merkmale der programmierbaren Maschine kennen gelernt haben, wollen wir nun versuchen, einige allgemeine Schlüsse für die praktische Arbeit aus den zugrunde liegenden Organisationsprinzipien zu ziehen. Wir hatten bisher festgestellt, dass der moderne Maschinenbegriff von der Realisierung der Maschine – also dem verwendeten Material und der konkreten Konstruktion – weitestgehend abstrahiert. Die Maschine ist damit ein symbolisches System, das durch seine Ein-/Ausgabebeziehungen, d.h. sein Verhalten definiert ist. Aber nicht nur die Maschine, auch die Objekte, die sie verarbeitet, sind symbolisch repräsentiert. Aus dem chiastischen Spiel zwischen Zeichen als Operanden und Zeichen als Operatoren erwächst im Nacheinander der Verarbeitung die Mächtigkeit des gesamten Systems. Ein zweites primäres Merkmal bilden die vor- und nachgeschalteten komplexen Interfaces. Ihr wesentliches Merkmal ist die Dissimulation. Durch die Interfaceprozesse wird die Zeichenhaftigkeit des Maschinenkerns wieder verdeckt. Wichtig ist auch die Erkenntnis, dass während der Entwicklung hin zur symbolischen Maschine die zugrunde liegenden Organisationsprinzipien weitestgehend unverändert geblieben sind, es sind die zur Psychostruktur sedimentierten historisch-gesellschaftlichen Strukturprinzipien. Diese archetypischen Prinzipien, die für die Strukturierung von Arbeitsprozessen zur Verfügung stehen, werden im Grunde seit Jahrtausenden angewendet. Sie finden sich in der antiken Megamaschine genauso wie im Handwerk oder der modernen Administration. Deshalb ist es nicht verwunderlich, dass wir sie auch in der programmierbaren Maschine wieder finden. Das Neue besteht darin, dass wir die Prinzipien als formale Einheiten anschreiben können. Die wesentlichen Prinzipien heißen:

- Sequenz von Arbeitsschritten,
- Wiederholung einer Sequenz (Schleife) zusammen mit jeweiliger Prüfung (Fallunterscheidung),
- Abstraktion,
- Modularisierung zusammen mit Hierarchien und Heterarchien.

Die ersten beiden dieser Prinzipien sind erforderlich, um sequentielle Abläufe universell zu beschreiben, sie bilden den Kern der klassischen Algorithmentheorie.

1 In: The Language of New Media, S. 48.

Abstraktionsfähigkeit ist nicht nur die Grundvoraussetzung jeder allgemeinen Erkenntnis, sondern insbesondere auch elementare Voraussetzung für die Entwicklung von Algorithmen. Die letzten Prinzipien – Modularisierung, Hierarchien und Heterarchien – sind allgemeine Methoden, um Komplexität in den Griff zu bekommen, beziehungsweise Komplexität zu erzeugen. Wir hatten gesehen, dass auf der Basis strikter Hierarchisierung sehr früh Megamaschinen realisiert werden konnten, deren Produktionen uns bis heute beeindrucken. Auch wenn bei sequentiellen Prozessen im Hinblick auf Turing-Universalität auf Hierarchien und Heterarchien verzichtet werden könnte, sind sie für die Strukturierung von komplexen Anwendungsprogrammen und damit für die Praxis von großer Bedeutung. Nicht nur Hardware und Programmiersprachen sind hierarchisch/heterarchisch strukturiert, im letzten Abschnitt hatten wir gesehen, dass die heutige hybride Maschine zum Beispiel in Form von Metasystemen ausgiebigen Gebrauch vom hierarchisch/heterachischen Prinzip macht und daraus großes Potential schöpft.

1.4.1 Programmierung als Zeichenprozess

„Man kann den Prozess, in dem etwas als Zeichen fungiert, Zeichenprozess oder Semiose nennen. Die Tradition, die bis auf die Griechen zurückgeht, stellt sich gemeinhin vor, dass dieser Prozess aus drei (oder vier) Faktoren bestehe: nämlich aus dem, was als Zeichen wirkt, aus dem, worauf das Zeichen referiert, und aus dem Effekt, der in irgendeinem Rezipienten ausgelöst wird und durch den die betreffende Sache ihm als Zeichen erscheint. Diese drei Komponenten der Semiose sollen jeweils Zeichenträger*,* Designat *und* Interpretant *heißen; hinzu kommt als vierter Faktor der* Interpret*.“*

Charles W. Morris[2]

Da Computer symbolische Maschinen sind, die sowohl aus Zeichen konstruiert, als auch durch Zeichen kontrolliert werden, können wir den Vorgang der Programmierung unter semiotischen Gesichtspunkten betrachten. Vom Begriff des Zeichens – von dem wir bisher bereits ausgiebig Gebrauch gemacht haben – hatten wir ein intuitives Verständnis vorausgesetzt. Vor allem aber hatten wir relevante Beispiele genannt, nämlich die gebräuchlichen Alphabete von Buchstaben und Ziffern, die für die bisherigen Betrachtungen vollkommen ausreichend waren. Der in der theoretischen Informatik verwendete Zeichenbegriff ist mathematisch – genauer mengentheoretisch – definiert und im Grunde von sehr einfacher Struktur. Grundeinheit ist ein Alphabet V, das aus einer endlichen Menge unterscheidbarer Zeichen besteht, also etwa V = {a, b, c}. Dieses einfache Verständnis des Zeichenbegriffs reicht normalerweise aus, um Automaten und symbolische Maschinen anzuschreiben und zu untersuchen. Auch die Deutung

2 In: Grundlagen der Zeichentheorie – Ästhetik der Zeichentheorie, S. 20.

von Eingabe-Zeichen einer realen Maschine lässt sich noch mit diesem intuitiven Begriff erfassen. Alle Einwirkungen der Umgebung auf eine Maschine in einem bestimmten Zustand, die darauf mit dem gleichen Folgeszustand reagiert, werden zu einem Zeichen zusammengefasst. Das heißt, Umweltereignisse, die sich so ähnlich sind, dass die Maschine sie nicht als verschiedene Einwirkungen auflösen kann, können durch ein einziges Zeichen markiert werden.

Die moderne Semiotik – die Theorie von den Zeichen –, die mit Namen wie Charles Sanders Peirce, Charles William Morris, Ferdinand de Saussure, Louis Hjelmslev und vielen anderen verbunden ist, hat im Vergleich zur Mathematik bzw. Informatik einen sehr viel komplexeren Zeichenbegriff entwickelt. Das Eingangszitat von Charles W. Morris vermittelt bereits eine gute Vorstellung vom semiotischen Zeichenbegriff. Wir wollen im Folgenden allerdings nicht die Definition von Morris, sondern den Peirceschen Zeichenbegriff für unsere Überlegungen zugrunde legen. Im 19. und 20. Jahrhundert verwendeten verschiedene Autoren das Bild des 'semiotischen Dreiecks', um die Beziehung zwischen Abbildfunktion und Interpretation eines Zeichens darzustellen. Leider werden für die Ecken dieses Dreiecks, und damit für die Grundbegriffe der Semiotik, sehr unterschiedliche Namen eingeführt – zum Teil sogar von den gleichen Autoren. Peirce verwendet die Begriffe 'Repesentamen', 'Objekt' und 'Interpretant', für die drei Faktoren des Zeichenbegriffs.[3]

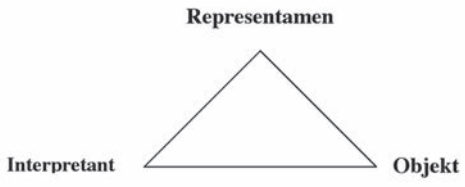


Abbildung 43:
Semiotisches Dreieck.

Nach Peirce[4] ist ein Zeichen etwas, das dazu dient, ein Wissen von einem anderen Ding zu vermitteln, das es vertritt oder darstellt. Dieses Ding nennt man Objekt des Zeichens. Die vom Zeichen hervorgerufene Idee im Geist, die ein geistiges Zeichen desselben Objekts ist, nennt man den Interpretanten des Zeichens. Um als Zeichen existieren zu können, sind also drei Konstituenten erforderlich:[5]

- Ein 'Representamen': Das Zeichen selbst, d.h. dasjenige, das steht. Zum Beispiel

3 Zum Beispiel: Interpretant (Peirce) = Sinn (Frege), Designatum (Morris), Begriff (Ferdinand de Saussure), Inhalt (Hjelmslev); Representamen (Peirce) = zeichenhaftes Vehikel (Morris), Ausdruck (Hjelmslev), Zeichen (Peirce); Objekt (Peirce) = Designat (Morris), Gegenstand (Frege). Vgl. Umberto Eco, Zeichen – Einführung in einen Begriff und seine Geschichte.

4 Vgl. Semiotische Schriften, Band 1, S. 204.

5 Peter Bøgh Andersen, A semiotic approach to programming, in: ders, et. al., The Computer as Medium, S. 17.

ein Bild, eine Geste, ein Klang, usw. Dafür muss es Eigenschaften besitzen, die es uns erlauben, es von anderen Zeichen zu unterscheiden.
- Ein 'Objekt': Das, wofür das Zeichen steht. Dieses Ding kann ein physikalisches Objekt, eine Klasse von Objekten, eine Emotion oder auch etwas Fiktionales sein.
- Ein 'Interpretant': Die Beziehung, in der das Representamen zum Objekt steht. Ein Interpretant kann als Reaktion eines Interpreters auf das Zeichen gesehen werden. Der Interpretant kann insbesondere selbst zum Zeichen werden, usw.

Zeichen lassen sich in drei Klassen einteilen, nämlich 'Icons', 'Indizes' und 'Symbole'. Ihre Unterscheidung wollen wir uns hier aus Platzgründen ersparen. In Anlehnung an Peter Bøgh Andersen können wir die Computerprogrammierung als semiotisches Netz darstellen.[6] Andersen stellt die Ausgangssituation für die Programmentwicklung als zwei miteinander verbundene semiotische Dreiecke dar. Da wir dem Interface in unseren vorangegangenen Überlegungen eine gesondere Stellung eingeräumt haben, ergibt sich aus unserer Sicht für die Programmierung, wie in der Abbildung zu sehen ist, ein aus mindestens drei Grundfiguren bestehendes semiotisches Netz.

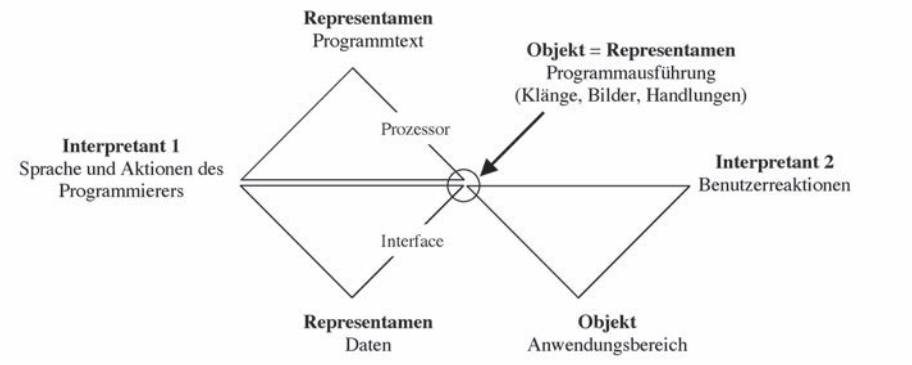


Abbildung 44:
Semiotisches Netz der Programmierung.

Bei computerbasierten Applikationen ist das Representamen das sichtbare, hörbare und fühlbare Resultat der Programmausführung. Das Objekt des Zeichenprozesses ist der Anwendungsbereich des Programms, zum Beispiel eine bestimmte künstlerische Installation, ein Computerspiel, ein Textverarbeitungsprozess, etc. Jeder Leser wird hier eigene Erfahrungen mit unterschiedlichen Anwendungsbereichen haben. Der Interpretant ist die Reaktion des Benutzers als Antwort auf die Programmausführung. Doch dies ist nur das im Bild rechte semiotische Dreieck eines Programms als

6 Ebd., S. 16 – 67.

Zeichenprozesses, nämlich jener Teil, den der Benutzer sieht. Dem Entwickler, der sich auf der Programmierebene bewegt, vermittelt sich ein anderes Bild. Der Programmtext zusammen mit der jeweiligen Entwicklungsumgebung ist hier das Representamen des Zeichenprozesses. Das Objekt, d.h. die Programmausführung, ist hierbei identisch mit dem Representamen des Benutzerprozesses. Was für den Entwickler Gegenstandsbereich der Anwendung ist, ist aus Sicht des Benutzers Representamen. Ein dritter semiotischer Prozess ist durch das Interface-Mapping gegeben. Die Daten, die der Prozess verarbeitet, werden dem Programm ebenfalls in Form von Representamen zur Verfügung gestellt. Der Zusammenhang zwischen dem Zeichenobjekt des Entwicklers, d.h. den sicht-, hör- und fühlbaren Ein-/Ausgaben der Programmverarbeitung und dem Representamen, d.h. den Daten des Programms, die als alphanumerische Zeichen repräsentiert sind, ist durch das Interface-Mapping realisiert. In vielen Fällen kennt der Programmierer gar nicht die genaue Wirkungsweise dieses Prozesses. Es ist aber auch nicht erforderlich, zum Beispiel zu wissen, wie eine digitale Kamera im Detail funktioniert. Entscheidend ist, dass der Programmierer das Format der numerischen Repräsentationen des Bildes kennt, in welchem die Bilddaten dem Programm zur Verfügung gestellt werden. In größeren Softwareentwicklungsprojekten wird das Interfacedesign als eigenständiges Problem betrachtet und von Spezialisten, etwa Mediengestaltern, realisiert. Die üblicherweise nur kognitiv geschulten Programmierer erhalten mit den in Wahrnehmung geschulten Gestaltern ihre komplementäre Ergänzung.

Was bezweckt nun die Darstellung der Programmierung als Semiose? Zum Einen lässt sie erkennen, wie komplex die Aufgabe des Programmierers für typische Anwendungen tatsächlich ist. Software ist durch eine 'Doppelexistenz' gekennzeichnet. Programme sind einerseits formalsprachliche Konstrukte, die nach syntaktischen Regeln aufgebaut sind, andererseits stehen Programme für Prozesse. Sobald das Programm auf einem Computer ausgeführt wird, schlägt der Text um in Verhalten. Während dem Anwender nur der Prozesscharakter der Software zugänglich ist, muss der Programmierer immer beides denken. Programmieren heißt gerade, den Programmtext so zu schreiben, dass das beabsichtigte Verhalten erzeugt wird. Der Softwareentwickler – im Bild Interpretant 1 – muss alle Komponenten des Prozesses im Blick haben, den Programmtext und die Darstellung der Daten genauso wie den Anwendungsbereich und die Reaktionen des Benutzers. Aber vor allem auch die im Zentrum des Diagramms stehende eigentliche Programmausführung. Im Ausführungsprozess, der die intendierten Handlungen, Bilder und Klänge erzeugt, laufen alle Überlegungen, die im Rahmen des Entwicklungsprozesses angestellt werden, zusammen und müssen sich bewähren. Die Apparate, auf denen die Representamen, d.h. die symbolischen Repräsentationen schließlich als Objekte dargestellt werden, wollen wir ganz allgemein als 'Displays' oder 'Interfaces' bezeichnen.

Das obige Diagramm ist in vielerlei Hinsicht unvollständig und ungenügend, dennoch zeigt es klar die Herausforderungen der Programmierung.

Ein Bildautor (Interpretant 1) wird sich zum Beispiel nicht nur Gedanken über die algorithmische Erzeugung des Bildes (Programmcode), die numerische Repräsentation (Daten, Auflösung) und die Präsentation des Bildes (Farben, Druckqualität, etc.) machen, sondern auch über den Kontext, innerhalb dessen das Bild und die Wirkung des Bildes auf den Betrachter entstehen. Es ergeben sich neue Gesichtspunkte, wenn wir einen Kurzschluss zwischen Interpretanten 1 und Interpretanten 2 herstellen, d.h. wenn beide ein und dieselbe Person sind, wie dies etwa der Fall ist, wenn Künstler oder Wissenschaftler ihre eigenen Werkzeuge programmieren. Der Entwickler wird dann in der Programmausführung seinem eigenen Denken, seinen eigenen Objektivationen in direkter Form wieder begegnen. In einem derartigen kurzgeschlossenen Prozess sind Systeme realisierbar, die sich weit jenseits ergonomischer Überlegungen entwickeln können. Kommerzielle Softwaresysteme sind Massenprodukte. Daraus folgt, dass die Individualität des Produktes weitestgehend ausgeschaltet wird zugunsten eines mittleren, gewissermaßen statistischen Benutzers, für den die Software möglichst reibungsfrei, d.h. ohne Bedien- und Systemfehler funktionieren soll. Künstler, die ihre eigenen Werkzeuge entwickeln, werden anders vorgehen. Programmsystem und Entwickler gehen hier eine sehr viel stärkere Bindung ein. Die Leistungsfähigkeit und Funktionalität des Systems wird sich hier in einem Prozess der gegenseitigen Adaption bewegen. Dieser Entwickler/Nutzer bewegt sich dabei im Raum seiner eigenen Gedanken, die er sukzessive auslotet und erweitert. Die Mechanismen eines derartigen hochgradig individuellen Entwicklungsprozesses, der in der Regel in eine Individualsoftware mündet, die nur noch vom Entwickler zu bedienen ist, ist nicht vergleichbar mit der Entwicklung von Software für den Massengebrauch, bei der die Entwicklung nur als notwendiges Übel und nicht als der wesentliche Prozess gesehen wird. Zum zweiten sind Programme als technische Systeme immer in einen gesellschaftlichen Kontext eingebunden. Neben diesen direkten Kurzschluss zwischen Nutzer und Entwickler ist Software – wie jedes technische Artefakt – in eine große Rückkopplungsschleife eingebunden. Terry Winograd und Fernando Flores beschreiben diesen Rückkopplungsmechanismus wie folgt:[7] „Jede neue Technologie entwickelt sich vor dem Hintergrund eines unausgesprochenen Verständnisses der Natur des Menschen und von menschlicher Arbeit. Der Umgang mit Technik wiederum führt zu fundamentalen Änderungen unserer Verhaltensweisen – und in letzter Instanz dazu, was es heißt, Mensch zu sein.“ Im Hinblick auf künstlerische und gestalterische Vorhaben lässt das Diagramm erkennen, dass im Zuge computerbasierter Medien die bisherigen Objekte, wie Bilder, Klänge, etc., vielschichtiger werden. Wir werden einige Konsequenzen, die aus der Darstellung der Programmierung als komplexer Zeichenprozess resultieren, im Folgenden genauer betrachten.

7 In: Erkenntnis Maschinen Verstehen, S. 11.

1.4.2 Medienobjekte

„Die Menschen waren nie Herren der Bilder. Sie können ebenso nie Herren der Zeichen werden. Die Mittel der Macht sind mächtiger als derjenige, der sie benutzt. "
Dietmar Kamper[8]

Medienobjekte bezeichnen in unserem Zusammenhang sehr viel mehr als die übliche Bedeutung in der Informatik, die mit dem Begriff des 'Objektes' vor allem auf den modularen Aufbau von Programmen mit Hilfe der Mechanismen 'Kapselung' und 'Vererbung' abzielt. Medienobjekte können einzelne Bilder sein, Klangobjekte, komplette 3D-Umgebungen, Webseiten, Datenbanken, künstlerische Installationen, etc. Verkürzt kann man sagen: Programmieren im künstlerischen Kontext heißt Medienobjekte herzustellen. Während programmiersprachliche Objekte der Informatik ihre Fragestellungen oft auf technisch-wissenschaftliche Gesichtspunkte reduzieren, sind künstlerische Medienobjekte enger gekoppelt an soziale und kulturelle Traditionen und die Vorstellung von Medien als Kommunikationsmittel. Für die im Folgenden beschriebenen Eigenschaften neuer Medienobjekte dient als Ausgangspunkt Lev Manovichs Gegenüberstellung neuer und alter Medien.[9] Manovich nennt fünf Prinzipien, durch die sich neue computerbasierte Medienobjekte von traditionellen Medienobjekten unterscheiden: numerische Repräsentation, Modularität, Automatisierung, Variabilität und Transcoding. Mit seinem Buch *The Language of New Media* beginnt Manovich eine in weiten Teilen noch ausstehende aber überaus wichtige Arbeit an den Grundlagen neuer Medien, indem er versucht, einige basale innere Strukturen computerbasierter Medien aufzudecken und ihre kulturelle Bedeutung herauszuarbeiten. Seine ersten beiden Prinzipien – numerische Repräsentation und Modularität – sind noch eng mit unseren eigenen Vorstellungen verbunden. Gleichzeitig fassen wir die numerische Repräsentation weiter und sehen hier in der Praxis bereits ein komplexes Schichtenmodell am Werke. Die Prinzipien, die Manovich unter dem Begriff Modularität zusammenfasst, fallen mit unserem Begriff der hierarchischen Organisation von Daten zusammen. Wir werden hier entsprechend den auch in der Literatur üblichen Begriff der Hierarchie verwenden. Insgesamt scheint Manovichs Klassifikation noch zu sehr dem klassischen Bild des Rechners verpflichtet zu bleiben. Er unterteilt strikt zwischen Daten und Prozessen und bezieht sich bei den Begriffen der Modularität (hier: hierarchische Organisation) und der numerischen Repräsentation vor allem auf die Daten. Erst mit Manovichs drittem Prinzip – der Automatisierung – kommen Verarbeitungsprozesse ins Spiel, deren Struktur allerdings wenig reflektiert wird. Unser Ziel ist es, neben den Strukturen der Daten medialer Objekte vor allem die Strukturen der Prozesse aufzuzeigen. Die derzeit in der Computerpraxis aufscheinenden und im letzten Abschnitt bereits ausführlich

8 Der Januskopf der Medien, Ästhetisierung der Wirklichkeit, Entrüstung der Sinne, S. 95 – 97.

9 Vgl. Principles of New Media, in: The Language of New Media, S. 27 – 48.

vorgestellten Prinzipien der Code-Reflexivitäten, d.h. der chiastischen Wechsel zwischen Daten und Prozessen und damit einhergehende spezielle Figuren wie Metasysteme, Hierarchien und Heterarchien, bleiben bei Manovich ausgeblendet. Erst durch die Code-Reflexivitäten und nicht durch die Datenstrukturen erhalten die Medienobjekte nach unserer Überzeugung ihre wirkliche Mächtigkeit. Wie bereits erläutert, kommt dabei im künstlerisch/medialen Kontext dem Interface-Mapping, durch das wichtige Abbildungsmechanismen zwischen Wahrnehmung und Kognition realisiert werden, sowie der freien Beweglichkeit der Codes (nicht nur der Daten) durch die Netze eine Sonderstellung zu.

1.4.2.1 Hierarchische Organisation von Daten

„Was computergestützte Bildproduktion von herkömmlichem Kunstschaffen unterscheidet, ist das enorme Optimierungspotential. Es kann jederzeit in alle Strukturebenen eines Werkes eingegriffen werden, um das Ergebnis immer wieder zu verbessern. Das Werk bleibt auf Dateiebene transparent und flexibel."
natürlich künstlich – Das virtuelle Bild[10]

Hierarchien sind – nicht nur im Bereich der Computer, sondern ganz generell – das mächtigste Organisationsprinzip zur Strukturierung komplexer Systeme. Die Hierarchisierung kann sich dabei sowohl auf Daten, als auch auf Prozesse (zum Beispiel Funktionalitäten von Subkomponenten) erstrecken. Manovich untersucht vor allem den hierarchischen Aufbau der Daten medialer Objekte.[11] Er stellt heraus, dass Medienobjekte (Bilder, Sounds, Formen, Verhalten, etc.) auf unterster Ebene Sammlungen diskreter Werte (Pixel, Polygone, Voxel, Buchstaben, etc.) sind. Diese Grundelemente werden über mehrere Stufen zu immer größeren Objekten zusammengefasst. Wichtig ist dabei, dass sie auf jeder Stufe ihre Identität behalten. Sie bewahren also bis zu einem gewissen Grad ihre Unabhängigkeit und sind jederzeit direkt ansprechbar, manipulierbar und auch austauschbar. Das hierarchische Strukturierungsprinzip hat im Bereich der Software sein Analogon bereits in den 1970-er Jahren gefunden. Zu dieser Zeit wurden Blockstrukturen und Unterprogramme eingeführt, um Software nicht nur übersichtlicher zu gestalten, sondern einzelne Elemente auch besser gegen unerwünschten Zugriff zu schützen und damit zuverlässigere Programme zu erhalten. Durch die Einführung von Unterprogrammen sind darüber hinaus erste Weichen in Richtung der heute sehr wichtigen Wiederverwendbarkeit von medialen Objekten gestellt worden. Allerdings ist festzuhalten, dass diese Programmstrukturierung nicht zu verwechseln ist mit der hierarchischen Organisation von Abläufen, die erst im vollen Umfang durchschlägt, wenn wirklich parallele Prozesse gestartet werden.

Jeder, der eigene Erfahrungen mit digitalen Bildern gesammelt hat, besitzt

10 In: Gerhard Mantz, et. al. (Hrsg.), Ausstellungskatalog zur Ausstellung „natürlich künstlich – Das virtuelle Bild", Buchrückentext.

11 Ebd., S. 30 ff.

eine Vorstellung von ihrem hierarchischen Aufbau. Wir wollen das Prinzip deshalb am nicht ganz so vertrauten Beispiel von Klang für alle Schichten aufzeigen. Der Computermusiker Curtis Roads sagt über die Bedeutung zeitlicher Hierarchien in der Musik:[12] „Music theory has long recognized a temporal hierarchy of structure in music compositions. A central task of composition has always been the management of interaction amongst structures on different time scales. Starting from the topmost layer and descending, one can dissect layers of structure, arriving at the bottom layer of individual notes. This hierarchy, however, is incomplete. Above the level of an individual piece are the cultural time spans defining the oeuvre of a composer or a stylistic period. Beneath the level of the note lies another multilayered stratum, the microsonic hierarchy. Like the quantum world of quarks, leptons, gluons, and bosons, the microsonic hierarchy was long invisible. Modern tools let us view and manipulate the microsonic layers from which all acoustic phenomena emerge. Beyond these physical time scales, mathematics defines two ideal temporal boundaries – the infinite and the infinitesimal – which appear in the theory of musical signal processing."

Curtis Roads macht deutlich, dass durch die Verwendung des Computers unterhalb der einzelnen Noten ganz neue, bisher nicht zugängliche Ebenen erschlossen werden. Beginnend mit der größten Skala, unterscheidet er neun Zeitskalen der Musik:

1. Infinit – ideale Zeitspannen von mathematischer (potentiell unendlich großer) Dauer, wie die unendliche Sinuswelle der klassischen Fourier-Analyse.
2. Supra – Zeitskalen jenseits der individuellen Komposition und ausgedehnt über Monate, Jahre, Jahrzehnte und Jahrhunderte.
3. Macro – Die Zeitskala der gesamten musikalischen Architektur oder Form, gemessen in Minuten oder Stunden, in Extremfällen Tagen.
4. Meso – Unterteilungen der Form. Gruppierungen von Klangobjekten in Hierarchien von unterschiedlich großen Phrasenstrukturen, gemessen in Minuten oder Sekunden.
5. Klangobjekt – Die Basiseinheit musikalischer Struktur, das traditionelle Konzept der Note verallgemeinernd, um komplexe und mutierende Klangereignisse auf der Zeitskala von einer bis mehrerer Sekunden einzubeziehen.
6. Micro – Klangpartikel auf der Zeitskala, die bis hinunter zum Schwellwert der Hörbarkeit reicht und in Millisekunden gemessen wird.
7. Sample – Die atomare Ebene digitaler Klangsysteme. Individuelle binäre Abtastwerte oder numerische Amplitudenwerte, die in starren Zeitintervallen aufeinander folgen.
8. Subsample – Fluktuationen auf einer Zeitskala im Nano-Sekundenbereich, die zu kurz sind, um richtig aufgezeichnet oder wahrgenommen zu werden.

12 In: Microsound, S. 3.

9. Infinitesimal – ideale Zeitspannen von mathematischer (potentiell unendlich kurzer) Dauer, wie die unendlich kurze Delta-Funktion.

Entscheidend ist, dass der Benutzer zu jeder Zeit Zugriff auf alle verwendeten Schichten der Daten hat.

1.4.2.2 Alphanumerische Repräsentation & Datenabstraktion

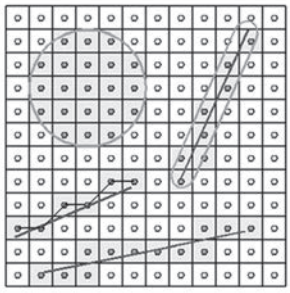
> *„Das Kunstwerk ist grundsätzlich immer reproduzierbar gewesen. Was Menschen gemacht hatten, das konnte auch nachgemacht werden. ... Dem gegenüber ist die technische Reproduktion des Kunstwerks etwas Neues, das sich in der Geschichte intermittierend, in weit auseinanderliegenden Schüben, aber mit wachsender Intensität durchsetzt. ... Man kann, was hier ausfällt, im Begriff der Aura zusammenfassen und sagen: was im Zeitalter der technischen Reproduzierbarkeit des Kunstwerks verkümmert, das ist seine Aura."*
>
> Walter Benjamin[13]

Auf der Ebene der alphanumerischen Repräsentation lassen sich Medienobjekte auf vollkommen unterschiedliche Weisen codieren. Ihre Representamen – um im oben eingeführten semiotischen Bild zu bleiben – können entweder numerisch, parametrisch oder funktional beschrieben werden.[14] Nehmen wir zum Beispiel ein Medienobjekt 'Bild'. Die einfachste Beschreibung eines Computerbildes ist das Rasterbild, d.h. die Aneinanderreihung der numerischen Werte für jeden Rasterpunkt (Pixel). Rasterbilder sind charakterisiert durch die Auflösung (Anzahl der Punkte pro Zentimeter oder Inch), die Datentiefe (Anzahl der Bit pro Farbkanal und Pixel eines Bildes) und den Farbmodus, d.h. das zugrunde liegende Farbmodell. Mit dem Begriff der numerischen Repräsentation zielt Manovich insbesondere auf diese Ebene der Datendarstellung ab. Vektorgrafiken sind dagegen bereits auf einer darüber liegenden Schicht angesiedelt. Da sie unabhängig von der Anzahl der Pixel sind, können sie auf jedem Ausgabegerät und bei jeder Skalierung scharf dargestellt werden. Vektorgrafiken sind charakterisiert durch eine Anzahl von Parametern für einen bestimmten Funktionstyp. Eine bekannte Funktionenklasse zur Darstellung von Kurven sind zum Beispiel Bezier-Polygone und B-splines, die anhand von Kontrollpunkten einfache rekursive Rekonstruktionen von Kurven erlauben. Gespeichert werden also nicht die Punkte der Kurve, sondern die Kontrollpunkte P_i. Die freie funktionale Beschreibung des Bildes setzt dagegen einen weiteren Abstraktionsprozess voraus. Hier können Funktionen nahezu beliebig spezifiziert werden.[15] Die funktionale Beschreibung liefert aber nicht nur genau ein Bild, sondern immer eine Klasse von Bildern. Der Parameterraum des Verfahrens spannt den Bildraum auf, der durch Variation der

13 In: Das Kunstwerk im Zeitalter seiner technischen Reproduzierbarkeit, in: ders., Medienästhetische Schriften, S. 352 – 355.

14 Siehe auch Zwischenspiel: Das Computerbild – Oder: Die Janusköpfigkeit des Pixels.

15 Siehe Zwischenspiel: Animation – Vom Thaumatrop über das Kino zum algorithmischen Bewegtbild.



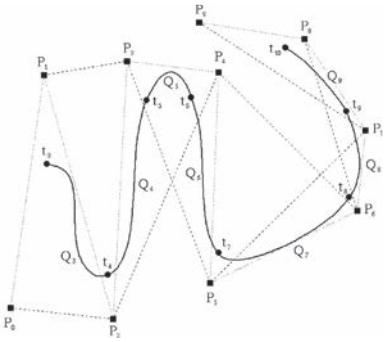


Abbildung 45/45a: Rasterbild (links), B-spline (rechts).

Parameter durchschritten werden kann.

Die Leistungsfähigkeit der funktionalen Beschreibung hängt wesentlich von den Möglichkeiten des Metawerkzeugs ab, mit dem die Funktionen spezifiziert und ausgewertet werden können. Ein Beispiel für ein sehr leistungsfähiges Werkzeug im Bereich der naturwissenschaftlichen Forschung ist Mathematica® der Firma Wolfram Research[16]. Einen anderen Weg geht PostScript®, das 1982 von der von John Warnock und Chuck Geschke gegründeten Firma Adobe Systems Inc. entwickelt wurde. PostScript ist eine spezielle Programmiersprache, mit der Bilder und Text hardwareunabhängig auf verschiedenen Ausgabeeinheiten wiedergegeben werden können. Eine PostScript-Datei besteht aus normalem ASCII Code, der mit jedem Texteditor lesbar ist. Der Code unterscheidet sich nicht wesentlich von höheren Programmiersprachen, d.h. er besteht aus Folgen von Anweisungen, Schleifen, Arrays, logischen Operatoren und Variablen. Auf diese Weise kann eine PostScript-Datei wie ein normales Programm erstellt und gelesen werden.

Zwei interessante Folgerungen können wir aus der skizzierten, auf unterschiedlichen Niveaus basierenden alphanumerischen Repräsentation von Medienobjekten ziehen:

1. Medien wie etwa Film, Video oder Druck sind oft Mischverhältnisse aus diskreten und stetigen Darstellungen. Ebenso sind auch die neuen Medienobjekte nicht notwendigerweise diskret, sondern in der Regel Verbindungen aus diskreten und stetigen Konzeptionen. Die so genannte digitale Technologie ist sehr wohl in der Lage, mit kontinuierlichen Objekten und exakten Beschreibungen von analogen Vorgängen umzugehen. Wir hatten gesehen, dass lediglich die niedrigste Ebene der Bildbeschreibung, die Rastergrafik, eine diskrete Bildbeschreibung darstellt. Die numerischen Werte stehen hier in direkter Weise für einzelne Bildpunkte. Parametrisierte und funktionale Beschreibungen sind dagegen sprachliche Konzeptionen, die auf ihrem jeweiligen Beschreibungsniveau sowohl diskrete als auch stetige Objekte modellieren können. Vektorgrafiken müssen demzufolge auch nicht zwangsweise als diskretes semiotisches Objekt realisiert, sondern können zum

16 Siehe http://www.wolfram.com/.

Beispiel als stetiges Objekt auf einem Plotter ausgegeben werden. Diskret sind lediglich die sprachlichen Beschreibungen der Objekte. Dies bedeutet aber keine computerimmanente Einschränkung, da Sprache – insbesondere geschriebene Sprache – immer aus diskreten Elementen zusammen gesetzt wird. Die Komplexität computerbasierter Medienobjekte ist demnach nicht begrenzt durch die Diskretheit der Beschreibung, sondern ausschließlich durch die Mächtigkeit der verwendeten Sprachen. Charakteristisch an den so genannten digitalen Medien ist demnach nicht die auf das Numerische fokusierende Digitalität, sondern die Zeichen- und Sprachbasiertheit.

2. Sprachlich beschriebene Medienobjekte können algorithmisch manipuliert werden. Die alphanumerische Repräsentation bildet nur eine Seite des Wesens der Objekte. Erst in Verbindung mit Algorithmen, die auf allen Ebenen der Repräsentation der Objekte ansetzen können, erhalten diese ihre Flexibilität und Struktur. Wie wir im Fall von PostScript gesehen haben, können alphanumerische Beschreibungen von Daten selbst wieder Programme sein, die von Postscript-fähigen Ausgabeeinheiten interpretiert werden können. Bereits auf der Ebene der Datenrepräsentation begegnen wir damit einem einfachen Beispiel der Code-Reflexivität. Aufgrund der alphanumerischen Beschreibung wird durch die Bearbeitung das 'Original' nicht notwendigerweise verändert oder zerstört. Es können leicht beliebig viele Versionen eines Objektes – oder Teile des Objektes – hergestellt, anschließend verändert, distribuiert und auch wieder spurlos beseitigt werden. Bei genauerer Betrachtung ist es schwer, die Rede vom Original im Zusammenhang mit den computerbasierten Medienobjekten aufrechtzuerhalten.

Nicht nur die Frage des Originals ist im Zusammenhang mit Medienobjekten schwierig, generell ist die Klärung der Identität eines Objektes ein nicht ganz leichtes Unterfangen. Der amerikanische Philosoph und Computerwissenschafter Brian Cantwell Smith arbeitet an einer Ontologie des Computers, um den vielfältigen Identitätsfragen medialer Objekte näher zu kommen. „Real-world computer systems involve extraordinarily complex issues of identity. Often, objects that for some purposes are best treated as unitary, single, or 'one', are for other purposes better distinguished, treated as several. Thus we have one program; but many copies. One procedure; many call sites. One call site; many executions. One product; many versions. One Web site; multiple servers. One url; several documents (also: several urls; one Web site). One file; several replicated copies (maybe synchronized). One function; several algorithms; myriad implementations. One variable; different values over time (as well as multiple variables; the same value). One login name; several users. And so on. Dealing with such identity questions is a recalcitrant issue that comes up in every corner of computing, from such relatively simple cases as Lisp's distinction between eq and equal to the (in general) undecidable question of whether two procedures compute the same function. The aim of the Computational Ontology project is to focus on identity as a technical problem

in its own right, and to develop a calculus of generalized object identity, one in which identity – the question of whether two entities are the same or different – is taken to be a dynamic and contextual matter of perspective, rather than a static or permanent fact about intrinsic structure."[17]

1.4.2.3 Datennetze

> *„By the end of the twentieth century, the problem was no longer how to create a new media object such as an image; the new problem was how to find an object that already exists somewhere."*
> Lev Manovich[18]

In Bezug auf Textanwendungen ist das signifikanteste Merkmal neuer medialer Organisationsformen die Hypertextualität, d.h. die Aufhebung der Linearität des Textes und die Verbindung ausgewählter Elemente des Textes durch Links. Der Begriff 'Hypertext' wurde in den 1960-er Jahren von Ted Nelson geprägt. In seinem ungewöhnlichen, von ihm 1974 selbst verlegten großformatigen Heft *Computer Lib/Dream Machines* sieht Nelson den Computer als hervorragendes Werkzeug, mit dem die tiefe Verschränktheit und gegenseitige Abhängigkeit aller Ideen und alles Wissens abgebildet werden kann. Durch Tim Berners-Lees World-Wide-Web-Protokoll wird Nelsons Idee des Hypertextes bis zu einem gewissen Grad realisiert (wohlgemerkt technisch), sein eigenes System *Xanadu* konnte dagegen nie breite Verbreitung erreichen. Hypermedia ist die konsequente Ausdehnung der informationellen Einheiten über reinen Text hinaus auf Einzel- und Bewegtbilder sowie den Audiobereich. Natürlich gab es Vorläufer dieses Prinzips bereits beim gedruckten Buch, etwa als einzelne Verweise oder ausgeprägt in Form der Enzyklopädie. Dennoch ist die klassische Struktur des Buches die Hierarchie. Die eigentlichen informationellen Einheiten befinden sich an den Blättern des Hierarchiebaumes und die darüber gelagerten Schichten der Unterabschnitte, Abschnitte und Kapitel bilden das Zugriffssystem. Hyperstrukturen sind hervorragend geeignet, Begriffe und Informationseinheiten in ihrem sprachlichen Bezug zu anderen Begriffen und anderen Wissenseinheiten darzustellen. Texte, die eine gewisse Dramaturgie, d.h. einen zeitlichen Aufbau innerhalb der linearen Rezeption besitzen, eignen sich dagegen weniger für die netzartige Darstellung, zumindest fehlen überzeugende Beispiele. Bei Hypertext und Hypermedia wird die bisherige Hierarchie der Daten durch das Netz ersetzt, beziehungsweise beide werden überlagert. Der Autor gibt hier allerdings noch in den meisten Fällen die Struktur des Netzes vor, erst die Rezeption, d.h. die Bewegung durch das Netz ist benutzergesteuert. Andere Ansätze wie die Internet Enzyklopädie Wikipedia[19] heben die Grenzen zwischen Autor und Leser auf,

17 In: Brian Cantwell Smith, On the Origin of Objects.

18 In: The Language of New Media, S. 35.

19 Siehe http://de.wikipedia.org.

indem jeder am Netzwerk der Begriffe und ihrer Erklärungen mitarbeiten kann.

Eines der grundlegenden Prinzipen der neuen Medienobjekte ist nach Manovich ihre Variabilität. Diese leitet er ab aus der numerischen Repräsentation, der Modularität und der Automatisierung. Unter Variabilität (Mutierbarkeit und Flüssigkeit) fasst er alle Eigenschaften zusammen, durch die die Rigidität bisheriger Medienobjekte bis zu einem gewissen Grad aufgehoben wird. Zwei Beispiele – Hypertext und Hypermedien – hatten wir bereits kennen gelernt. Unterschiedliche Pfade durch das Dokument sind aus Sicht des Nutzers gewissermaßen unterschiedliche Versionen eines Dokumentes. Weitere Beispiele sind die Skalierbarkeit der Daten und automatisierte Updates, die die Aktualität der Objekte sicher stellen. Eng verbunden mit dem Problem der Datenorganisation ist das Problem der Datenarchivierung und des Zugriffs auf diese Daten. Datenbanken sind nicht nur Systeme, die es erlauben, riesige Mengen von Daten in unterschiedlicher Form und Auflösung zu speichern und verfügbar zu machen, sie bieten auch vielfältige Möglichkeiten der Organisation und des Zugriffs auf die Daten an. Durch vorgeschaltete Interfaces und Zugriffsmuster können die Inhalte individuell zusammengestellt und aufbereitet werden. Durch 'kollaborative Filter' kann das System Präferenzen eines Benutzers – etwa beim Kauf von Büchern – in Beziehung zu anderen Nutzern stellen. Das System benutzt dieses sich ständig verändernde Beziehungsgeflecht zum Beispiel, um dem Käufer scheinbar individuelle Vorschläge zu unterbreiten. Manovich vertritt die These, dass die Variabilität der neuen Medienobjekte mit der postindustriellen Logik des 'production on demand' und 'just in time' korrespondiert. Medien sollen individueller und für jeden Benutzer maßgeschneidert sein. Hierzu werden – ebenfalls automatisiert – Benutzerprofile und Modelle angelegt, die aus dem Verhalten des Benutzers erstellt werden. Die bisher besprochenen Formen der hierarchischen und vernetzten Organisation bezogen sich ausschließlich auf Daten. Die Methoden des Filterns, Konvertierens und Modellierens von Daten gehören dagegen bereits zu den Prozessen der Medienobjekte.

1.4.2.4 Die Prozessstruktur medialer Objekte

Eine Stärke von Hypermedien liegt in der Trennung zwischen der Netzstruktur und den Inhalten. Gewählte Datenformate, sowohl für die Beschreibung des Netzes als auch der Daten selbst, sind bis zu einem gewissen Grad willkürlich und werden vom Systementwickler aufgrund relativ subjektiver Argumente festgelegt. Unterschiedliche Programme arbeiten deshalb häufig aufgrund fehlender Standards mit stark abweichenden Datenformaten. Um die Austauschbarkeit zwischen Systemen zu gewährleisten, werden deshalb in jedem Werkzeug Funktionen angeboten – so genannte 'Converter' – die zwischen den verschiedenen Formaten übersetzen. Während die Syntax durch diesen Vorgang meist vollständig umgekrempelt wird, bleibt die Semantik des Objektes unberührt. 'Filter' sind eine weitere Klasse von Programmen, die in engem

Zusammenhang mit Medienobjekten stehen. Jedes Bildverarbeitungssystem bietet Standardoperationen an, die es erlauben, die Qualität eines Bildes zu verbessern oder zu verändern. Jeder geschulte Bildoperateur kennt die Wirkungsweisen dieser Operationen und wendet sie – zumeist mehrere nacheinander – auf die Bilddaten an, um bestimmte Effekte zu erreichen. Die oben angesprochenen kollaborativen Filter sind noch komplexere Programmvarianten, die Verbindungen zwischen Benutzern herstellen und so versuchen, sogar soziale Parameter zu erfassen. Die neuen computerbasierten Medienobjekte bestehen damit nicht nur aus visuellen und/oder akustischen Phänomenen und den dazugehörigen symbolischen Beschreibungen, die auf unterschiedlichen Abstraktionsstufen vorliegen können, sondern zusätzlich aus Algorithmen, die auf diese Beschreibungen einwirken, sie interpretieren und umformen. Lev Manovich bezeichnet diese Grundeigenschaft neuer Medienobjekte als 'Automatisierung'. Sicherlich sind Automatismen, die zu einer gekoppelten und verschränkten Verarbeitung der Daten durch den Benutzer und das System führen, ein wesentliches Merkmal computerbasierter Objekte. Die Stärke der automatisierten Verarbeitung kann sich aber erst voll entfalten, wenn Code-Reflexivitäten ins Spiel kommen.

1.4.2.5 Medienobjekte und Code-Reflexivitäten

Automatismen, die Medienobjekte bearbeiten, können unterschiedlich komplex sein. Konverter und Filter sind relativ einfache Vertreter. Die im letzten Abschnitt behandelten Agenten sind dagegen am oberen Ende der Skala angesiedelt. Da sie weitestgehend eigenständig auf andere Medienobjekte einwirken oder mit ihnen kommunizieren, entziehen sie sich der direkten Kontrolle des Benutzers. Die derzeit fortgeschrittensten Beispiele autonomer Medienobjekte sind wahrscheinlich Charaktere in Computerspielen. Sie sind nicht nur Experten für bestimmte Aufgaben innerhalb eines größeren Medienobjektes – des Computerspiels – und determinieren damit wesentlich den Verlauf des Spiels, sie sind darüber hinaus lernfähig und verändern während des Spiels autonom ihre eigenen Verhaltensweisen. Agenten oder lernfähige Avatare sind gleichzeitig avancierte Beispiele für Code-Reflexivitäten, die Merkmale von selbstreferenziellen Strukturen, von Metastrukturen oder auch von Heterarchien besitzen können. Als zwei sehr einfache Beispiele für die Code-Reflexivität hatten wir Web Browser und die Programmiersprache PostScript genannt. Web Browser transportieren Java-Applets als Daten durch das Netz, um sie auf dem Zielrechner als Prozesse zu starten. Die Programmiersprache PostScript erlaubt es, Druckdateien auflösungs- und geräteunabhängig zu beschreiben. Das setzt voraus, dass Programme in der Lage sind, als Ausgabedaten Programme zu schreiben, die von anderen Prozessoren ausgeführt werden können.

Nach Lev Manovich können neue Medienobjekte generell als aus zwei Schichten bestehend gedacht werden. „The structure of the computer image is a case in point. On the level of representation, it belongs to the side of

human culture, automatically entering in dialog with other images, other cultural 'semes' and 'mythemes'. But on another level, it is a computer file that consists of a machine-readable header, followed by numbers representing color values of its pixels. On this level it enters into a dialog with other computer files. The dimensions of this dialog are the image's content, meanings, or formal qualities, but rather file size, file type, type of compression used, file format, and so on."[20] Als Beispiele für die kulturelle Schicht nennt Manovich Enzyklopädien, Kurzgeschichten, Kompositionen, Mimesis und Katharsis, sowie Komödie und Tragödie. Datenpakete, Dateien, Funktionen und Variablen, Listen und Felder, sowie Sortier- und Vergleichsverfahren sind dagegen in seiner Sichtweise Beispiele für die 'Computerebene'. Er folgert daraus, dass neue Medien aus einer kulturellen Ebene und einer Computerebene bestehen. Aus dieser Perspektive scheint die Computerschicht in keinerlei Tradition zu stehen und muss demnach vollkommen neu sein. Nach Lektüre unseres Abschnitts 1.2 zur „Struktur der programmierbaren Maschine" sollte allerdings klar sein, dass die im Computer realisierten Elementarstrukturen ebenfalls auf eine sehr lange Tradition zurückblicken können. Es wäre deshalb richtiger festzustellen, dass in den neuen Medienobjekten kulturelle Modelle aufeinander treffen, die bisher in weitestgehend getrennten Feldern entwickelt wurden und sich durch ihre Verbindung neue Möglichkeiten eröffnen. Durch diese Vereinigung und gegenseitige Einflussnahme entstehen nicht nur neue Formate und kulturelle Formen, sondern auch neue technische und wissenschaftliche Herausforderungen.

Zusammenfassend lässt sich feststellen: Die neuen Medienobjekte sind keine singulären Erscheinungen – keine Originale –, die fest an ein bestimmtes Material gebunden sind und sich nach ihrer Produktion lediglich durch materielle Alterungsprozesse verändern, sondern Komplexionen aus hierarchisch und heterarchisch organisierten Daten und Prozessen, die in ein komplexes Wechselspiel der gegenseitigen Beeinflussung verstrickt sind. Die Daten sind zusammengesetzt aus alphanumerischen Beschreibungen, die unterschiedliche Abstraktionsstufen repräsentieren und deren syntaktische Form zwischen verschiedenen Darstellungen wechseln kann. Diesen Wechsel zwischen verschiedenen Darstellungen von Objekten bewerkstelligen Converter, die zum festen Erscheinungsbild im Umgang mit Medien geworden sind. Große Bedeutung im Zusammenhang mit computerbasierten Medienobjekten besitzen Archive und der automatisierte Zugriff auf die Medieninhalte. Verbunden mit den Objekten sind immer Automatismen, die einzelne Objekte generieren, manipulieren, variieren und in neue Verhältnisse zueinander stellen. Es gibt typische Problemstellungen, die in unterschiedlichen Kontexten immer wieder auftauchen. Ein wesentliches Kennzeichen der neuen Objekte ist ihre Wiederverwendbarkeit, die sich bis auf die Code-Ebene niederschlägt und zu Medienobjekten führt, denen der

20 In: The Language of New Media, S. 46 f.

Programmierer in verschiedenen Kontexten immer wieder begegnen wird und die er deshalb immer wieder verwenden kann. Im künstlerischen Kontext verbinden Medienobjekte zwei kulturelle Traditionen miteinander: Die Kulturgeschichte der Maschine und die Kunstgeschichte. Programmieren heißt, diese Ebenen zu integrieren und permanent Übersetzungsarbeit in beide Richtungen zu leisten.

1.4.3 Interfaces – Zwischen Wahrnehmung und Kognition

„Die Abbildtheorie der Repräsentation wird also schon zu Beginn durch ihr Unvermögen behindert, zu spezifizieren, was kopiert werden soll. Nicht ein Gegenstand in der Weise, wie er ist, noch in allen Weisen, noch in der Weise, wie er für das geistige Auge aussieht. Darüber hinaus ist gerade an der Vorstellung etwas verkehrt, irgendeine der Weisen, in der ein Gegenstand ist, irgendeinen Aspekt von ihm, zu kopieren. Denn ein Aspekt ist nicht nur der Gegenstand-aus-einer-gegebenen-Entfernung-und-einem-Blickwinkel-und-in-gegebener-Beleuchtung; er ist der Gegenstand, wie wir ihn betrachten oder begreifen, eine Version oder ein Konstrukt des Gegenstandes. Wenn wir einen Gegenstand repräsentieren, dann kopieren wir nicht solch ein Konstrukt oder eine Interpretation – wir stellen sie her.“

Nelson Goodman[21]

Die Automatismen der neuen Medienobjekte, die in der Prozesshaftigkeit der Maschine gründen, entfalten vor allem im Zusammenspiel mit Interfaces ihr Potential. Digitale Technologien werden zu Mechanismen der automatischen Bild-, Klang- und Handlungsproduktion. Die neuen Medienobjekte sind mit einer relativen Autonomie gegenüber den Benutzern ausgestattet und in der Lage, nicht nur Zeit zu akkumulieren, sondern sie zu stauchen, zu strecken, neu zu organisieren und zu erzeugen. Interfaces sind deshalb immer auch vor dem Hintergrund ihrer Eigenschaft zu bewerten, wie sie mit Zeit umgehen. Unsere semiotischen Betrachtungen der Programmierung benutzten indirekt den Begriff der Repräsentation. Das Peircesche Representamen steht für etwas Anderes (das Objekt), das für jemand Dritten (den Interpreten) Bedeutung hat. Wichtig ist jedoch festzuhalten, dass es dabei nicht um die Repräsentation einer Außenwelt, also einer Spaltung der Welt in eine objektive Realität und sein Modell davon im Computer geht, sondern um Repräsentation einer bereits konstruierten Welt, d.h. einer Welt, die bereits von Interpreten mit Semantik belegt ist. Wenn unsere Anwendung zum Beispiel aus der Welt des ‘Internet Bankings’ oder der ‘Flugbuchung’ stammt, dann ist die Beziehung zwischen Repesentamen und Objekt nicht die zwischen Realität und Begriff, sondern das Objekt selbst (Bankkonto oder Flugbuchung) sind bereits kognitive Objektivationen. Computer-Interfaces als Mittler zwischen System und Umgebung sind immer Verbindungen zwischen

21 In: Sprachen der Kunst.

den inneren Strukturen der Maschine und gewissen allgemeinen Vorstellungen und Bedeutungen, die eine Gemeinschaft von Benutzern und Programmierern sich vom Anwendungsbereich machen. Diese Vorstellungen sind also bereits Ergebnisse eines Objektivierungsprozesses. Eine Ausnahme bilden die von Peter Cariani beschriebenen Systeme, die selbständig neue semantische Kategorien finden. Als funktionsfähige Technologie existieren solche Systeme bisher allerdings nicht.

1.4.3.1 Tasten, Schalter, Regler

Interfaces haben eingabeseitig die Aufgabe, Ereignisse aus der Umgebung des Computers in syntaktische Darstellungen umzuwandeln. Die Tastatur ist eine direkte Umsetzung dieser Aufgabe. Die syntaktischen Zeichen, mit denen die Maschine intern arbeitet, werden hier auf der Oberfläche der Interfaces aufgebracht und können als Zeichen in die Maschine eingegeben werden, indem die entsprechende Taste gedrückt wird. Mit aufeinander folgenden Tastendrücken ist zugleich die zeitliche und räumliche Linerarisierung der Ereignisse verbunden. Buchstaben, deren Tasten später gedrückt werden, erscheinen in der Regel in einer Datei weiter hinten oder werden später verarbeitet. Wir haben keinerlei Möglichkeit zu prüfen, ob die gewünschten Zeichen wirklich in der Maschine repräsentiert sind. Wir vertrauen dem Interface insoweit, als wir davon ausgehen, dass die gewollten Zeichen wirklich gespeichert werden, sobald sie auch auf dem Bildschirm erscheinen. Aber befinden wir uns hier nicht in der Situation des Philosophen René Descartes, der in seinen *Meditationen über die Grundlagen der Philosophie* die Frage stellt, wie wir unseren Sinnen trauen können und ob nicht vielleicht alles eine Illusion ist, die von einem mächtigen Dämon (einem Hacker) geschaffen wurde, um uns zu täuschen? Tatsächlich haben wir auf dieser Ebene der Kommunikation mit der Maschine – und auch auf der Ebene der Programmierung – keine Möglichkeit, dies in letzter Konsequenz auszuschließen. Der normale Benutzer wird deshalb die gleiche Haltung gegenüber dem Computer einnehmen, die er auch zu jedem anderen Artefakt im Laufe der Zeit entwickelt. Solange das Artefakt sich hinreichend oft so verhält, wie man es von ihm erwartet, wird sich die Frage nicht stellen und der Benutzer wird ein gewisses Vertrauen in die Technologie entwickeln. In der Wirtschaft wird dieses Problem des Descartesschen Dämons aber angesichts weltweiter Hackeraktivitäten ernsthaft diskutiert. Unter dem Begriff 'Trusted Computing' werden erste Lösungsversuche angeboten.[22]

Tastaturen, aber auch Knöpfe, Schalter, Dreh- und Schieberegler, sind direkte Verbindungen zu den alphanumerischen Repräsentationen im Rechner. Diese Interfaces sind sehr nah an den Konzepten der Maschine und ihre Benutzung bezieht sich in der Programmierung auf alphanumerische Repräsentationen. Während Tastaturen vor allem auf die Eingabe von Texten abzielen, sind

22 Vgl. Fred von Lohmann, Meditationen über Trusted Computing, in: CODE – The Language of Our Time, S. 46 – 50.

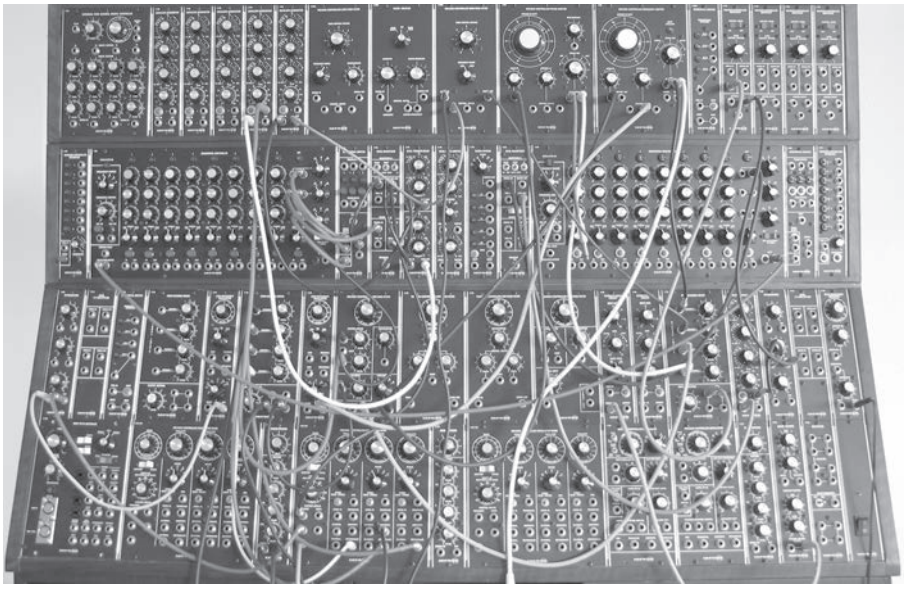


Abbildung 46:
Gerd Peun, *Analogium* (2004). Interface des analogen Sound-Synthesizers. Knöpfe, Schalter und Regler sind als Interfaces sehr nahe an der technischen Architektur und repräsentieren steuerbare Systemparameter. In programmierbaren, digitalen Systemen sind Knöpfe, Schalter und Regler direkt verbunden mit Programmvariablen.

Schalter und Knöpfe logische Elemente, Dreh- und Schieberegler numerische Eingabeeinheiten. Allen diesen Interfaces ist gemein, dass sie direkt mit internen Zuständen der Maschine verbunden sind. Was die Eingaben im Einzelnen bedeuten und welche Wirkung sie auslösen, hängt vor allem vom Kontext ab, in dem die einzelnen Werte auftreten. Tastaturen, Knöpfe und Regler sind längst zu Kulturtechniken geworden, die bereits jedes Kind bedienen kann und dessen Funktionen es schon in den ersten Lebensjahren spielerisch begreifen lernt. Es ist deshalb nur konsequent, wenn auch im Zusammenhang mit dem Computer diese Schnittstellen als generische Einheiten eingesetzt werden. Die allgemein bekannte Funktionsweise dieser Bedienelemente muss in der Bedienung mit dem jeweils aktuellen Kontext in Verbindung gebracht werden. Erst durch diese Trennung zwischen abstrakter Funktionsweise (Schalter als logische Entscheidung zwischen zwei Zuständen, etc.) und tatsächlicher Funktion wird es möglich, die komplexen und vielfältigen Anwendungen des Computers auf minimale Standardschnittstellen abzubilden.

Die Leistungsfähigkeit dieses Prinzips wird konsequent in heutigen Video- und Computerspielen genutzt. Electronic Arts, weltgrößter Videospielhersteller, der unter anderem das Spiel *FIFA*, ein Fußballsimulationsspiel, auf den Markt gebracht hat, verwendet komplexe Abbildungen zwischen den Aktionen des Spielers und ihren Auswirkungen auf das Spiel. Mit einfachen Knöpfen und Joysticks werden komplexe Abläufe kontrolliert. Bewegungen von Profis werden

hierzu mit 'Motion Capture Technologien' aufgezeichnet und können durch bestimmte Tastenkombinationen ausgelöst und überzeugend rekombiniert werden. Die Aktionen des Spielers vermischen sich dabei mit den Eigenaktionen des Spiels. Besonders der vollkommen unerfahrende Videogame-Spieler wird auf diese Weise in ansprechende und komplexe Spielsituationen versetzt. Durch das komplexe Interface-Mapping werden seine relativ groben eigenen Aktionen in gelungene Spielzüge überführt. Numerische Werte werden hier durch aufwendige Verfahren auf komplexe Handlungen abgebildet.

1.4.3.2 Komplexe Synthesen und Analysen

„Die neuen digitalen Technologien nehmen uns nicht den Körper, sondern unsere Illusionen bezüglich des Geistes und seiner Kräfte. ... Sie simulieren nicht das Reale, sondern erinnern uns, dass wir schon immer nicht-existente Dinge zum Handeln benutzen (Punkte, Linien, Zeichen, Begriffe, Namen)."

Maurzio Lazzarato[23]

In der Regel verläuft die Komplexitätstransformation des Interface-Mappings jedoch in entgegen gesetzter Richtung. Es werden nicht einfache Benutzerentscheidungen in komplexe Handlungen umgewandelt, sondern eine komplexe Umgebung wird auf einfache numerische Skalen reduziert. Nehmen wir als einfaches Beispiel ein Thermometer. Aus der Vielfalt des Umweltgeschehens in das ein Thermometer eingebettet ist, wird ein einziger numerischer Wert isoliert: die Temperatur am Messfühler. Dies ist einerseits eine drastische Reduktion der Umwelt auf einen einzigen Wert, gleichzeitig ist es eine Erweiterung der eben beschriebenen Spielkonsole, da die für die Verarbeitung notwendigen Zeichen nicht von Hand über Tastaturen, Regler und Joysticks einzugeben sind, sondern maschinell erzeugt werden. Eingabeseitig findet durch das Interface-Mapping eine semantische Determinierung und gleichzeitig die syntaktische Codierung statt. Aus der Umwelt werden – dem Bauprinzip des Interfaces entsprechend – ganz bestimmte Ereignisse ausgewählt und mit Bedeutung belegt. Diese bedeutsamen Ereignisse werden numerisch codiert und können anschließend digital verarbeitet werden. Welche Ereignisse bedeutsam sein sollen, entscheidet der Konstrukteur des Interfaces. Der Begriff der Temperatur ist eine komplexe wissenschaftliche Abstraktion, nämlich die Reduktion thermodynamischer Vorgänge auf einen einzigen Punkt einer mathematischen Werteskala. Im Bauplan des Interfaces ist somit immer schon eine Modellvorstellung der Wirklichkeit implementiert. Wichtig ist hierbei, dass dieser Prozess des Interface-Mappings nie vollständig digital sein kann, sondern zumindest in der ersten Stufe analoge physikalische, chemische, biologische, etc. Prozesse mit einschließen muss. Wenn wir diese Prozesse als analog bezeichnen, so folgt daraus nicht, dass sie sich einer

23 In: Videophilosophie – Zeitwahrnehmung im Postfordismus, S. 12.

mechanischen Beschreibung entziehen. Viele chemische und auch biologische Vorgänge werden in den Naturwissenschaften sogar äußerst mechanistisch erklärt. Es heißt lediglich, dass diese Prozesse zumindest teilweise in einem stetigen, nicht-symbolischen Raum stattfinden. Dieser nichtsprachliche Bereich erfordert deshalb nicht in allen Belangen eine Objektivierung, es reicht, wenn das Interface die gewünschten Daten liefert. Im weiteren Verlauf können die generierten Signale komplexen digitalen Verarbeitungsprozessen unterzogen werden, bevor sie als Zeichen die Maschine erreichen. Wichtig ist, dass es bei der programmierbaren Maschine nicht um die Differenz von Analogem und Digitalem geht. Es geht im Interface vielmehr um die sprachliche Determinierung und zeichenbasierte Codierung von Handlungen und Phänomenen durch benutzerdefinierte Verfahren. Dieser modellbasierte Vorgang ist folglich nur über die Modellbildungsprozesse der Entwickler und Benutzer zu verstehen. Die Beziehung zwischen der Realität und unserer Vorstellung davon ist immer modellvermittelt, darüber hinaus gibt es bei rechnerbasierten Prozessen keinen Bezug zur Realität. An dieser Stelle wollen wir auf das typische Verständnis von Realität, wie es vor allem in den Technikwissenschaften vertreten wird, näher eingehen. Wir werden sehen, dass die Sichtweise für den nicht-technischen Umgang mit Medienobjekten erweitert werden muss.

Die wissenschaftliche Metapher, auf die sich z.B. die Entwicklung der digitalen Bildtechnologien bezieht, ist im wesentlichen die Vorstellung eines 'Eindrucks', den ein Objekt auf einem lichtempfindlichen Träger hinterlässt. Sehen wird als optische Wahrnehmung definiert, die Welt und Bild einander gegenüberstellt. Diese Arbeitshypothese ist durchaus mächtig und dient unter anderem in der wissenschaftlichen Computergrafik als Arbeitsmodell für die Herstellung vollkommen synthetischer, so genannter photorealistischer Bildwelten. Aufgrund der Gegenüberstellung von Welt und Bild und der Abbildfunktion des Modells endet hier aber auch bereits das Potenzial der Computergrafik. Ihr fehlt eine Theorie der Kreation eines Bildes, die über dieses physikalische Abbildungsmodell und seine sensorische Wahrnehmung hinausgeht. Die Produktion synthetischer Bilder muss jedoch nicht prinzipiell im Abbildungsmodell gefangen bleiben, sondern kann auf vollkommen anderen Konzeptionen von Objekten, Räumen und Bildvorstellungen beruhen. Die Bedeutung der neuen Technologien liegt nach dieser Vorstellung nicht in der Fähigkeit, Welt aufzuzeichnen, sondern in den ihr eigenen Möglichkeiten, imaginierte Welten ins Werk zu setzen. Daraus folgt insbesondere – und dies hat auch Bedeutung für technische Bildsyntheseprozesse, für den künstlerischen Umgang und die Herstellung von Medienobjekten –, dass die intellektuelle Arbeit der Bildproduktion nicht auf optische, akustische, taktile oder olfaktorische Modelle reduziert werden darf. Die in den Interface-Apparaten (Kameras, Mikrophone, Motion Tracking Devices, Bildschirme, Lautsprecher, etc.) wirksamen Abbildungen, die eingabeseitig bedeutungsvolle Ereignisse und Gegebenheiten der Umgebung auf syntaktische Zeichen reduzieren und

ausgabeseitig Zeichen mit Bedeutung aufladen, sind nur auf den untersten Repräsentationsebenen bedeutsam. Sie bilden immer nur die erste Schicht der Gesamtkonzeption und müssen ergänzt werden durch die eigentlichen Vorstellungen von Bild, Klang, Bewegung, etc. Der konzeptuelle Raum beginnt gewissermaßen erst über der Pixelebene. Die Betrachtung des Interfaces macht somit immer nur im Hinblick auf die Architektur der gesamten Arbeit Sinn. Auf Programmierebene mit Bildern und Klängen zu arbeiten, heißt immer, in einem konzeptionellen Raum zu arbeiten. Zwischen die Ebene der syntaktischen Codes und der eigentlichen Wahrnehmung einer Arbeit ist immer eine methodische Ebene eingezogen. Statistische Verfahren, Datenbankkonzepte, Bildverarbeitungsalgorithmen oder spezielle Lern- und Optimierungsmethoden wie neuronale Netze und genetische Programmierung können in dieser Hinsicht für künstlerische Arbeiten genauso wichtig werden wie vordem Wahrnehmungsprinzipien. Hinter der ästhetischen Aufbereitung der Eingabedaten sowie der Ausgabendaten müssen die eigentlichen Konzepte und Vorstellungen stehen, die sowohl für die Auswahl der Interface-Apparatur als auch für die Entwicklung des die Daten verarbeitenden Programms entscheidend sind. Diese neuen Bild- und Klangproduktionen kann man nur verstehen, wenn man die wahrnehmungsorientierten Modelle von Auge und Ohr ergänzt durch kognitionsorientierte Modelle der Wissenschaften und der Informatik. Die beiden Abbildungen zu den Arbeiten *Inoutsite II* von Ursula

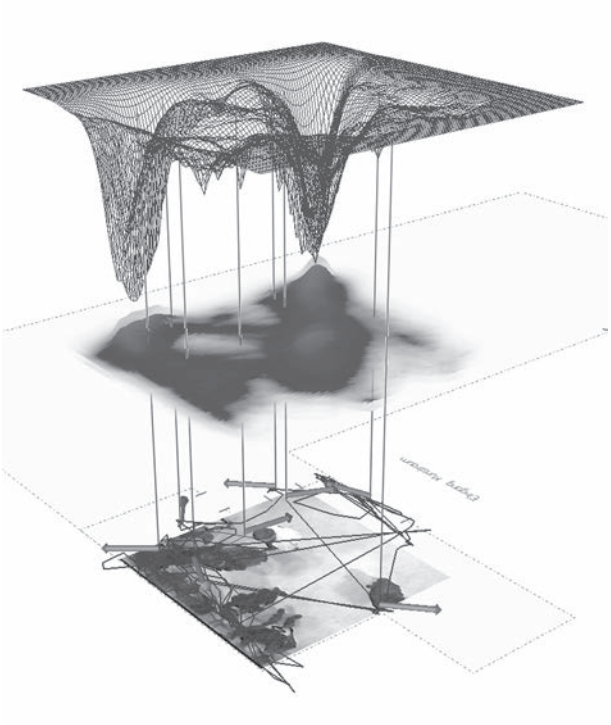


Abbildung 47:
Ursula Damm, *Inoutsite II.*
Eine Kamera wird über einem öffentlichen Platz so angebracht, dass ein möglichst aviatisches Bild des Geschehens auf dem Platz entsteht. Aus dem Videobild werden im ersten Schritt die sich bewegenden Objekte des Bildes herausgefiltert. Dabei wird versucht, die einzelnen sich bewegenden Personen voneinander zu unterscheiden und ihre Wege zu erfassen. In der grafischen Darstellung wird vorrangig die sich permanent verschiebende Nutzung des Raums dargestellt. Durch die Überlagerung des Original-Videobildes mit den virtuellen Grafiken erhalten die Vorbeigehenden die Möglichkeit, selbst mit dem Bild zu spielen und Sounds und Geometrien zu produzieren, die das Ergebnis des eigenen Verhaltens sind (siehe auch Projekt-Steckbriefe).

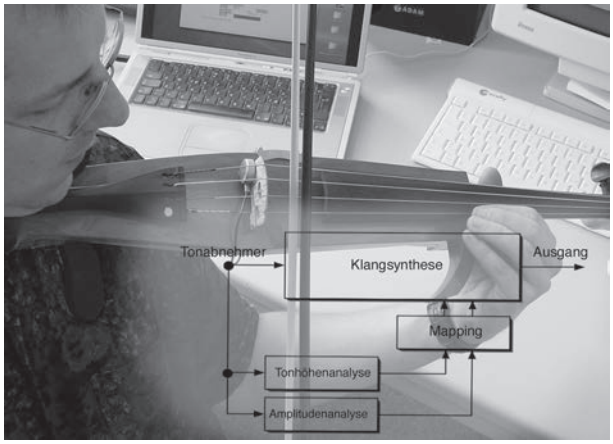


Abbildung 48:
Cornelius Pöpel, *Schwingungsgesteuerte Klangsynthese*. Im Bereich der Computermusik kommt innerhalb der so genannten Sound-Interfaces der Frage der Spielbarkeit eines Instruments eine große Bedeutung zu. Das auf dem Prinzip der schwingungsgesteuerten Klangsynthese basierende System ermöglicht es, synthetische Klänge streicherspezifisch spielbar zu machen. Über ein einfaches Pickup wird das Audiosignal einer E-Bratsche abgenommen, digitalisiert und einem in Max/MSP implementierten DSP-Algorithmus zugeführt. Der Klangsyntheseprozess ist so aufgebaut, dass er primär vom digitalisierten Audiosignal gesteuert und von Analyseparametern wie Tonhöhe und Amplitude modifiziert wird. Klangliche Nuancen jenseits der analysierten Parameter sind dadurch trotzdem im Ausgangssignal des Systems repräsentiert. Die eingesetzten Algorithmen bilden somit den neuen Klangkörper des Instruments. So wie der Bratschist sich vorher die klanglichen Qualitäten des analogen Instruments erarbeiten musste, lernt er nun gewissermaßen 'die Algorithmen zu spielen'.[24]

Damm und *Schwingungsgesteuerte Klangsynthese* von Cornelius Pöpel unterstützen das Gesagte. Die unmittelbare Wahrnehmungsebene von Rasterbildern oder von digitalen Klängen ist immer im Zusammenhang mit den Methoden des konzeptuellen Raums zu betrachten.

1.4.4 Zum Verhältnis von Autor und Werk

„The machines are restless tonight."
Alluquere Rosanne Stone[25]

Wir wollen den Zusammenhang zwischen formaler Beschreibung eines Werks und dem Erlebnis des Rezipienten – der in allgemeiner Form mit dem semiotischen Netz der Programmierung am Beginn des Abschnitts diskutiert wurde – noch einmal im Hinblick auf das Dreiecksverhältnis von Autor, Werk und Rezipient betrachten. Jeder Autor, der eine formale Beschreibung einer interaktiven Anwendung erstellt, muss verschiedene Abstraktions- und Codierleistungen erbringen. Die große Herausforderung interaktiver Anwendungen im

24 Vgl., Cornelius Pöpel, Synthesized Strings for Stringplayers, Hamamatsu 2004.

25 In: Will the Real Body Please Stand Up? Boundary Stories about Virtual Cultures, in: Michael Benedikt (Hrsg.), Cyberspace: First Steps, S. 81.

künstlerischen Umfeld ist die Inszenierung des Geschehens. Der Autor muss dabei nicht nur in der Lage sein, die Prozesse und Abläufe, die entstehen, wenn sein Programm auf einem Rechner ausgeführt und auf einem Display zur Darstellung gebracht wird, bis zu einem bestimmten Grad zu antizipieren, sondern sie auch dramaturgisch und ästhetisch inszenieren. Die besondere Herausforderung künstlerischer Computeranwendungen besteht darin, dass im besonderen Maße Verstand und Sinnlichkeit, Gefühl und Vernunft verschmolzen werden müssen, und das auf der Basis einer bisher hauptsächlich der rationalistischen Tradition verpflichteten Maschine. Der Autor muss dabei nicht nur die enorm schwierige Formalisierungsleistung vollbringen, die zu kommunizierende Idee auf einen syntaktischen Prozess zu reduzieren, sondern er muss auch das Interface definieren und bei interaktiven Anwendungen zusätzlich die Eingaben des Rezipienten antizipieren. Erlaubt dürfen dabei nur Benutzereingriffe sein, die die Inszenierung nicht zerstören. Die auf syntaktischer Ebene stattfindenden Benutzermanipulationen müssen deshalb auf ihre semantische Auswirkung hin vorgedacht werden. 'Erfahren' kann die Arbeit letztlich nur auf der 'Performativen Ebene' werden, d.h. indem der lineare und zeitlose Programmtext von einem Rechner ausgeführt wird. Als Rechenprozess erhält die Anwendung ihren zeitlichen Charakter zurück und die Ergebnisse können auf einem geeigneten Display (Bildschirm, Lautsprecher, etc.) zur Darstellung gebracht werden. Durch den laufenden Interaktionsprozess kommt für den Rezipienten (Interakteur) jene Unschärfe zurück ins Geschehen, die im Vorgang der Formalisierung, d.h. der Reduktion auf die syntaktische Beschreibung, eliminiert wurde. Das Wechselspiel zwischen Rezipienten und Werk, das auf der performativen Ebene stattfindet, kann im Fall computerbasierter Arbeiten durch den Begriff der 'Improvisation' charakterisiert werden. Unter Improvisation wollen wir die Performance eines Benutzers innerhalb eines durch einen Autor gesetzten Rahmens verstehen. Abhängig von den gewählten Regeln kann der Rahmen im Hinblick auf den Verlauf der Erzählung sehr eng oder sehr weit gesteckt sein. Das Konzept der Improvisation impliziert weitere grundlegende Prinzipien: 1. Der Rezipient kennt die Grundregeln der Interaktion und synchronisiert sich durch seine Handlungen mit dem vorgegebenen Rahmen. 2. Der konkrete Verlauf ergibt sich als komplexes Zusammenspiel zwischen starren Regeln und Handlungsfreiheiten des Benutzers.

Im Hinblick auf mediale Anwendungen tritt in der Programmierung zunehmend ein weiterer Aspekt in den Vordergrund: die Fähigkeit von Software, autonomes Verhalten zu generieren. Hierbei sind unterschiedliche Grade von Autonomie zu unterscheiden. Wird der Begriff strikt angewendet, dann gibt es derzeit keine autonomen Programme. Zwei strukturelle Bedingungen für Autonomie, wie wir sie lebenden Systemen zuschreiben, sind operationelle Geschlossenheit und vollständige Selbstbezüglichkeit. Wir hatten bei unseren Betrachtungen zu Code-Reflexivitäten und Interfaces gesehen, dass beides auf der Basis gegenwärtiger Modelle nicht gegeben ist. Beim Autonomiegedanken geht

es darum, Wahlfreiheiten zu erzeugen, um die Vielfalt und Multiperspektivität zu erhöhen und die Stringenz des Handelns zurückzudrängen.

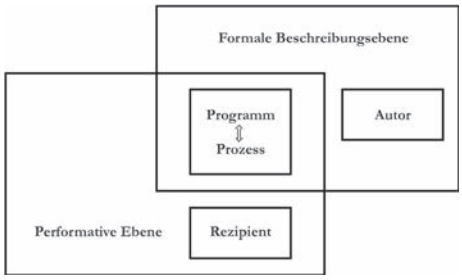


Abbildung 49:
Deskriptive und performative Ebene computerbasierter Werke.

„Die Geschichte der Menschen und die Evolution der Natur könnten dadurch beschrieben werden, dass sie ein Arsenal von Mechaniken hervorbringen, 'Mechaniken, die über den Mechanismus und Automatismus triumphieren' und so die 'Möglichkeit einer Wahl' freisetzen. Um sich zu entfalten, braucht die affektive Kraft der Zeit eine Mechanik, die das Bewusstsein von der zweckbestimmten Handlung befreit. Diese Mechaniken vergrößern die Fähigkeit, Unbestimmtheit (Unvorhersehbares) in Materie und Geist einzufügen. Die elektronischen und digitalen Technologien setzen in ganz spezifischer Weise eine 'Möglichkeit von Wahl' und Aktion frei, denn sie imitieren intellektuelle Arbeit und befreien sie gewissermaßen."[26] In den meisten künstlerischen Arbeiten, die bereits diese Form von Handlungsfreiheit und Autonomie für ihre Software postulieren, wird der Begriff nur oberflächlich verwendet und die Systeme sind von autonomem Verhalten weit entfernt. Häufig handelt es sich um Systeme, bei denen gewisse Anfangsbedingungen gegeben sind, und eine Reihe von Regeln, nach denen das System sich von hier aus entwickelt. Tatsächlich können solche Systeme schwer durchschaubar sein, d.h. um herauszufinden, wie das System sich wirklich verhält, bleibt keine andere Möglichkeit, als es laufen zu lassen und zu beobachten. Wir werden diese Systeme im nächsten Abschnitt unter dem Gesichtspunkt der Offenheit eingehender betrachten. Derzeitige Ansätze, denen im Programmablauf tatsächlich eine gewisse Autonomie gegenüber dem Autor zugesprochen werden muss, sind Systeme, die mit Lernverfahren und KI-Modulen arbeiten. Auch hier hat die Spielindustrie eine Vorreiterrolle übernommen. Sobald der Programmcode sich durch den Verlauf der Interaktion selbst modifiziert, erhalten wir die nächste Stufe der Ablösung des Codes vom Programmierer. Aber auch diese Systeme können nach Gotthard Günther noch keine eigenständigen Entscheidungen treffen. Für ihn sind dies immer noch Pseudoentscheidungen, deren Alternativen vollständig im konzeptuellen Bereich des Programmierers liegen. Um von 'echten' Entscheidungen sprechen zu können, fordert Günther die maschinelle Fähigkeit der 'Selbsterzeugung von Wahlmöglichkeiten'. „On the other hand, a machine, capable of genuine decision-making, would be a system gifted with the power of self-generation of choices, and the acting in a decisional manner upon its self-created alternatives. (...) A machine, which has such a capacity could either accept

26 Maurizio Lazzarato, a.a.O., S. 104 f.

or reject the total conceptual range within which a given input is logically and mathematically located. It goes without saying that by rejecting it the machine displays some independence from the programmer which would mean that the machine has the logical and mathematical prerequisites of making decisions of its own which were not implied by the conceptual range of the program."[27] Dies wäre die weitreichendste Form der Autonomie, die auch identisch ist mit Peter Carianis emergenten semantischen Funktionen, d.h. Systemen, die fähig sind, selbständig neue semantische Kategorien auszubilden. Neben der Machbarkeit bleibt vor allem die Frage zu beantworten, wieweit solche Systeme wünschenswert sind.

Betont werden muss, dass die Frage der Komplexität einer Software und ihre Fähigkeit, autonomes Verhalten zu zeigen, keinerlei Qualitätskriterium für künstlerische Werke sein kann. Dennoch wird bei computerbasierten Arbeiten dieser Aspekt des Eigenanteils und des Eigenverhaltens der Maschine gerade von Künstlern oft als wichtig herausgestellt. Klar ist, dass die Autonomie einer Software und die Eigendynamik ihres Verhaltens die Rezeption einer Arbeit in der Regel stark beeinflusst. Die erreichte Autonomie hängt stark davon ab, welcher Grad an Komplexität implementiert wird. Der Begriff der Komplexität soll hierbei zunächst vom Begriff der Kompliziertheit unterschieden werden. John von Neumann verwendet im Rahmen seiner Untersuchungen zu selbstreproduzierenden Systemen den Begriff der Kompliziertheit[28] um zu zeigen, dass ein minimales Niveau an Relationen zwischen den Komponenten eines Systems gegeben sein muss, damit es fähig ist, sich selbst zu reproduzieren. Im Allgemeinen ist Kompliziertheit ein Maß für die Anzahl der Subsysteme und Parameter, die notwendig sind, um ein System vollständig zu beschreiben. Gelingt die formale Beschreibung, dann kann das System auf der Basis dieser Beschreibung vollständig verstanden werden. Der Begriff der Komplexität eines Systems dagegen bezeichnet die Schwierigkeit, das Verhalten eines Systems innerhalb eines einzigen Bezugsrahmens vollständig zu erfassen, selbst wenn man die komplette Information über alle Komponenten besitzt. In dieser Hinsicht können bereits sehr einfache formale Systeme große Komplexität besitzen. Ein einfaches Mittel, um Komplexität zu erreichen, sind Code-Reflexivitäten.

Im Hinblick auf das Verhältnis von Autor und Werk lässt sich festhalten, dass durch die Eigenaktivität computerbasierter Medienobjekte neue Formen der Autorschaft entstehen. Hierbei sind verschiedene Stufen der Ablösung des Werks von seinem Entwickler zu unterscheiden. Dazu drei Beispiele:

1. Die niedrigste Stufe, Systemvariationen und Wahlmöglichkeiten für den Rezipienten zu schaffen, sind Hyperstrukturen. In jedem Knoten des Netzes hat der Nutzer neue Entscheidungsmöglichkeiten, wobei das gesamte Netz und damit alle möglichen Pfade vollständig vom Autor festgelegt werden.

27 In: Proposal for the Continuation of an Investigation of a Mathematical System for Decision-Making Machines, S. 6.

28 Siehe auch Zwischenspiel: Zelluläre Automaten & Selbstreproduktion.

Alle Wahlmöglichkeiten sind hier während des Systementwurfs explizit vom Autor antizipiert, deshalb wird die Struktur des Netzes in der Regel nach einigem Umgang mit dem System transparent werden. Dies entspricht dem herkömmlichen Verhältnis zwischen Autor und Werk. Auch das Vorsehen aleatorischer Elemente würde noch zu dieser Klasse der Autorschaft zählen.

2. Die nächste Stufe sind nicht-triviale Automaten im Sinne Heinz von Foersters.[29] Durch Angabe von Anfangsbedingungen und einer Reihe von Regeln wird ein formaler Rahmen aufgespannt. Eingaben aus der Umgebung bewirken Zustandsübergänge des Systems, wobei aufgrund seines 'Gedächtnisses' gleiche Eingaben zu verschiedenen Zeitpunkten unterschiedliche Systemreaktionen bewirken. Das Gedächtnis wird durch Variablen realisiert, deren Werte für die Bestimmung von Folgezuständen mit berücksichtigt werden. Solche Systeme sind für denjenigen, der den inneren Mechanismus nicht kennt, von außen nicht mehr analytisch determinierbar. Der Autor definiert hier den Rahmen, innerhalb dessen der Rezipient sich bewegen kann. Dieser ist aber nicht in der Lage, die innere Struktur des Systems vollständig aufzudecken. Aufgrund seines Gedächtnisses zeigt das System zeitabhängiges Verhalten.
3. Die dritte Stufe schließlich sind Code-Reflexivitäten. Das System wird hier seine eigene innere Struktur im Laufe der Interaktion umbauen und selbständig weiterentwickeln. In „Seele und Maschine" sagt Gotthard Günther bereits 1956 Maschinen vorher, deren Arbeitsweise uns in wesentlichen Hinsichten unbekannt sein wird. Hierbei bezieht er sich in erster Linie auf die selbstreproduzierenden Automaten John von Neumanns. „Der Mensch sieht sich also schließlich Maschinen gegenüber, die er selbst weder gebaut noch im strengen Sinne selber entworfen hat. Es ist dabei durchaus möglich, dass er sich schließlich Maschinentypen gegenüber sieht, die er nicht mehr ganz versteht und deren Ableitung von dem von ihm selbst entworfenen Originaltyp immer schwieriger wird, weil mit der fortschreitenden Generationenfolge die Zahl der Willkürelemente, die die Entwicklung beeinflusst haben, ständig wächst."[30] Diese Maschinen sind in Form von lernenden Systemen, genetischen Algorithmen und selbstreproduzierenden Automaten ansatzweise Realität.

Aus Sicht der Autoren sind – je nach Anwendungsgebiet – alle drei Verfahren interessant. Sofern das System gut inszeniert wird, können auch festverdrahtete Entscheidungen einer Hyperstruktur vollkommen überzeugendes Verhalten generieren. Die Frage der Autonomie eines Systems ist eng gekoppelt an eine andere wichtige Qualität von Systemen: die Offenheit. Der Begriff der Autonomie soll hier explizit auf das Verhältnis von Autor und Werk abzielen, insbesondere auf die Frage, durch welche Codestruktur die sukzessive Ablösung des Codes von seinem Entwickler unterstützt wird. Offenheit einer Software fragt dagegen danach, wie sich die Verhältnisse aus der Sicht des Rezipienten darstellen.

29 Ebd.

30 In: Beiträge zur Grundlegung einer operationsfähigen Dialektik, Band 1, S. 83.

Natürlich gibt es auf der Ebene der Codes Überschneidungen zwischen den Sichtweisen.

1.4.5 Offenheit – Wesentliches bleibt unformalisiert

„Die Poetik des 'offenen' Kunstwerks strebt, wie Pousseur sagt, danach, im Interpreten 'Akte bewusster Freiheit' hervorzurufen, ihn zum aktiven Zentrum eines Netzwerkes von unausgeschöpften Beziehungen zu machen, unter denen er seine Form herstellt, ohne von einer Notwendigkeit bestimmt zu sein, die ihm die definitiven Modi der Organisation des interpretierten Kunstwerks vorschriebe; doch könnte man einwenden, dass jedes Kunstwerk, auch wenn es nicht als materiell unabgeschlossen erscheint, eine freie und schöpferische Antwort fordert, schon deshalb, weil es nicht wirklich verstanden werden kann, wenn der Interpretierende es nicht in einem Akt der Kongenialität mit seinem Urheber neu erfindet. Allerdings ist dies eine Erkenntnis, zu der die zeitgenössische Ästhetik erst nach dem Erlangen einer reifen kritischen Bewusstheit hinsichtlich der interpretativen Beziehung gekommen ist; frühere Künstler waren von dieser Bewusstheit noch weit entfernt."

Umberto Eco[31]

Das Zitat von Umberto Eco zum 'offenen Kunstwerk' macht deutlich, dass das Verhältnis zwischen Autor und Rezipient nie ein einfaches Sender-Empfänger-Verhältnis ist, sondern immer bis zu einem gewissen Grad 'Co-Creation' voraussetzt. Vieles, das ganz allgemein über Kunstwerke gesagt werden kann, gilt unabhängig vom gewählten Medium und damit gleichermaßen für traditionelle wie für computerbasierte Arbeiten. Für formale Programmstrukturen gelten dennoch in bestimmten Hinsichten besondere Verhältnisse. Wir wollen kurz in Erinnerung rufen, welchen Bedingungen jede Formalisierung unterliegt. Ein Vorgang ist dann formal beschreibbar, wenn er die Bedingungen der Schriftlichkeit, der Schematisierbarkeit und der Interpretationsfreiheit erfüllt.[32] Schriftlichkeit bedeutet, dass wir den Vorgang mit Hilfe von Zeichen ausdrücken können. Schematisierbarkeit heißt, der Ablauf lässt sich als festes Schema, als allgemeines Verfahren (d.h. algorithmisch) beschreiben, und Interpretationsfreiheit meint, dass wir vollkommen frei in der Wahl der Zeichen und Bezeichnungen (Variablennamen, etc.) sind, da wir bei der Ausführung der Operationen keinen Bezug mehr darauf nehmen, wofür die Zeichen stehen. Durch den Vorgang der Formalisierung entledigen wir uns auf der Ebene der Zeichen jeder Bedeutung. Die Zeichen – Programme wie Daten – können sich im Prozess der Verarbeitung nur auf sich selbst beziehen, es existiert nichts sonst, worauf sie Bezug nehmen könnten. Der Außenstehende kann nur dann die Bedeutung aus den Zeichen

31 In: Das offene Kunstwerk, S. 31 f.

32 Vgl. Sybille Krämer, Symbolische Maschinen, S. 1 – 4.

lesen, wenn er den Schlüssel der Codierung kennt. Alle drei Bedingungen der Formalisierung sind automatisch erfüllt, wenn es gelingt, einen Ablauf in einer üblichen Programmiersprache auszudrücken. Formale Strukturen – so die allgemeine Vorstellung – lassen der Imagination keinen Raum. Durch die Formalisierung soll die Intention ihres Autors exakt erfasst und jede Missdeutung und Unschärfe ausgeschaltet werden. Nun zeichnen sich künstlerische Arbeiten aber dadurch aus, dass kein Werk dem Betrachter genau das mitteilen kann, was der Künstler beabsichtigt. Voraussetzung für die echte Partizipation des Betrachters ist, dass jedes Werk Lesarten ermöglicht, die von der Intention des Entwicklers und seinen Formalisierungen abweichen. Es stellt sich die Frage, wo beim technischen Medium Computer der Freiraum für die Partizipation und die Imagination des Betrachters und die Unschärfen der Programmierung liegen. Erlauben die formalen Grundmuster des Computers und seiner Programmierung lediglich unzweideutige Vermittlungen zwischen Autor und Rezipienten? Das Ziel der Formalisierung besteht gerade darin, jeden Interpretationsspielraum auszuräumen und den Formalismus möglichst eng an Intentionen des Ausführenden zu koppeln. Gibt es dennoch Freiräume, die zwischen dem Programmcode noch Nebeneffekte jenseits der bewussten Setzung zulassen? Sind auf der Basis des Computers überhaupt offene Werke im Sinne Ecos möglich?

Unter Offenheit einer Software wollen wir ganz allgemein die Eigenschaft verstehen, dem Benutzer bzw. Rezipienten jeweils neue Handlungs- und Interpretationsspielräume zu eröffnen und eine gewisse Flexibilität und Durchlässigkeit für dessen Intentionen zu schaffen. Um diese Form von Offenheit zu erreichen – so unsere These – dürfen tatsächlich wesentliche Qualitäten des Systems nicht formalisiert werden. Der Programmtext ist aber immer das explizit und unzweideutig Ausgedrückte. Auf dieser Beschreibungsebene können Freiheiten nur erreicht werden, indem wir die bereits häufig diskutierten Selbstveränderungsstrategien für Codes einsetzen, d.h. Lernverfahren, Selbstreferenzen, bzw. allgemein Code-Reflexivitäten. Nur indem der Code sich selbst fortschreibt, kann auf der performativen Ebene des Codes Neues entstehen, das auch aus der Sicht des Autors als neu gewertet werden muss. Wir wollen im Folgenden weitere Quellen betrachten, die aus der Sicht des Rezipienten ebenfalls zur Offenheit von Programmen beitragen.

1.4.5.1 Abstraktion

„Psychologen wie etwa Miller haben Experimente durchgeführt, und festgestellt, dass die maximale Anzahl an Informationseinheiten, die ein Mensch gleichzeitig verarbeiten kann, in einem Bereich von sieben plus/minus zwei liegt. Die eingeschränkte Aufnahmefähigkeit scheint mit dem Leistungsvermögen des Kurzzeitgedächtnisses zusammenzuhängen. ... Damit sind wir mit einem fundamentalen Dilemma konfrontiert. Die Komplexität der Software-Systeme, die wir entwickeln wollen, steigt beständig, obwohl unsere Fähigkeit, mit dieser Komplexität umzugehen, relativ beschränkt ist. Wie können wir uns aus dieser Zwangslage befreien?“

Grady Booch[33]

Wie oben ausgeführt, liegt die Bedeutung der neuen Technologien nicht in der Fähigkeit, Welt aufzuzeichnen und wieder darzustellen, sondern in den ihr eigenen Möglichkeiten, imaginierte Welten zu realisieren und intellektuelle Anstrengung zu simulieren. Bestimmte Denkstrukturen können entäußert und an die Maschine übergeben werden. Durch den Ablauf auf der Maschine begegnen wir diesen Gedankenfolgen als etwas Eigenständigem wieder. Da Programme implementierte Theorien sind, kann in der Ausführung des Programms der Vorgang der Abstraktion, der während der Theoriebildung stattgefunden hat, umgekehrt werden, d.h. wir begegnen in der Simulation den Phänomenen wieder, die Ausgang der Generalisierung und Theoriebildung waren. Das Abstrakte ist – zumindest umgangsprachlich – das Gegenteil des Konkreten. Abstraktion ist dementsprechend der Vorgang, bei dem sich Beschreibungen von den konkreten Erscheinungen lösen. Durch diesen Abstand von den konkreten Dingen muss die Abstraktion dem Uneingeweihten als bedeutungslos erscheinen. Entscheidend ist aber, dass wir im experimentellen Umgang mit solchen Systemen das Konkrete zurückholen und sogar neue Phänomene erzeugen können, die wir bisher so nicht wahrgenommen hatten, die aber ebenfalls im Möglichkeitsraum der Theorie liegen. Nehmen wir als Beispiel die Newtonsche Mechanik.[34] Das 2. Newtonsche Gesetz ist eine enorme Verdichtung von Einzelwahrnehmungen in einem einzigen Differentialgleichungssystem. Das Prinzip ist in der Lage, zugleich makroskopische wie mikrosopische Bewegungsabläufe zu erfassen und damit die himmlische und die irdische Physik zusammenzuführen. Durch Implementierung des 2. Newtonschen Gesetzes auf der Basis entsprechender numerischer Lösungsverfahren können aus diesem Prinzip alle Phänomene der Bewegung gewissermaßen wieder entfaltet werden. Das Prinzip der Abstraktion und die anschließende Umkehrung des Prozesses, indem durch konkrete Wahl von Parametern und Anfangsbedingungen eine Vielheit von Phänomenen erzeugt wird, ist ein mächtiges Prinzip, um offene Systeme zu erzeugen. Dabei muss nicht, wie beim Newtonschen Gesetz, ein

33 In: Objektorientierte Analyse und Design – Mit praktischen Anwendungsbeispielen, S. 31 f.

34 Siehe auch Zwischenspiel: Die Weltmaschine.

natürliches Vorbild existieren, sondern das formale System kann vollkommen fiktiv gewählt werden. In der Computerspielentwicklung hat man beispielsweise festgestellt, dass antizipative Systeme ab einer gewissen Komplexität des Spiels nicht mehr realisierbar sind. Bei hunderttausenden von Spielern wird es vermehrt vorkommen, dass sich Spieler Lösungen überlegen, die im konkreten Spielkontext vollkommen plausibel sind, die aber von den Entwicklern einfach nicht vorgesehen wurden. Ein Killerkriterium für jedes Spiel! Spielentwickler gehen deshalb verstärkt dazu über, nicht mehr alle Situationen im Detail zu antizipieren, sondern ein kleines regelbasiertes 'Universum' zu implementieren, innerhalb dessen sich alle möglichen Lösungswege als natürliche Systemzustände erweisen. Solche Systeme sind durchaus in der Lage, ihren eigenen Autor zu überraschen, auch wenn das System sich im Rahmen eines vorher aufgespannten Raums von Möglichkeiten bewegt. Wenn wir Naturwissenschaft als Versuch auffassen, von speziellen Erscheinungen zu allgemeinen Gesetzen zu kommen, dann besteht die Entwurfsaufgabe für offene Systeme darin, diesen Prozess umzukehren, d.h. man muss allgemeine Regeln so wählen, dass die daraus resultierenden Phänomene die gewünschten Qualitäten besitzen. Entscheidend ist, dass durch die Abstraktion die Qualitäten, die der Betrachter wahrnimmt, aus der Formalisierung verschwinden. Die Qualitäten einer Bewegung, ihre Ästhetik, Form, Dauer, etc., sind nicht mehr im Formalismus selbst, d.h. im Programm zu finden, sondern nur noch in den Randbedingungen, sprich der Parameterwahl. Wenn man den Betrachter bittet, die Qualität einer Bewegung zu beschreiben, wird er dies mit anderen Begriffen tun als denen, die auf Programmebene verwendet werden. Während der Betrachter vielleicht von einem Saldo spricht, oder von Anmut, der Eleganz oder Hektik einer Bewegung, sind auf Implementierungsebene lediglich Variablen zu finden, die Masse, Kraft, Beschleunigung, Ort und numerische Lösungsverfahren charakterisieren. Diese Kluft zwischen den Beschreibungsebenen ist ein generelles Merkmal offener Systeme. In dieser Hinsicht sind Bankkontenverwaltungen und Flugbuchungssysteme geschlossene Systeme, bei denen direkte Zusammenhänge zwischen den Qualitäten des Systems und seiner Implementierung existieren.

1.4.5.2 Verschränkung paralleler Aktivitäten – Heterarchien

Eine weitere Quelle für die Offenheit von Systemen sind heterarchische Verschränkungen von gleichzeitig aktiven Einheiten. In der Literatur wird die Eigenschaft von parallelen Systemen, auf der Beobachtungsebene Phänomene zu erzeugen, die auf der Beschreibungsebene nicht repräsentiert sind, auch als 'Emergenz' bezeichnet. Cariani[35] unterscheidet drei Modelle emergenten Verhaltens: 'Computational Emergence', 'Thermodynamic Emergence' und 'Emergence Relative to a Model'. Bei der informatischen Emergenz ergibt sich das interessierende globale Verhalten aus der Interaktion lokaler Regeln. Fraktale

35 Vgl. Emergence and Artificial Life, In: Christopher Langton, et. al., Artificial Life II, S. 775 – 797.

und zelluläre Automaten[36] sind einschlägige und allgemein bekannte Beispiele für solche Mikroheterarchien, die dem Betrachter auf der Wahrnehmungs- und Interaktionsebene eine Vielfalt an Handlungs- und Interpretationsspielräumen eröffnen. Das globale Verhalten lässt sich hierbei weder direkt aus den lokalen Regeln ablesen noch lässt es sich durch ausschließlich analytische Betrachtungen vollständig vorhersagen. Das System selbst weiss also nichts von den globalen Strukturen, es braucht einen Beobachter außerhalb des Systems, der die globalen Strukturen wahrnimmt. Im Hinblick auf den Entwurf solcher Systeme gibt es verschiedene Probleme. Erstens lassen sich keine Anleitungen angeben, wie das System zu konstruieren ist, damit sich ein vom Entwickler gewünschtes Verhalten einstellt. Zweitens gibt es keine Rückkopplung von der globalen Ebene auf die niedrigere Ebene der Regeln. Die höhere Ebene nimmt keinen Einfluss auf die lokalen Regeln, wie dies beispielsweise in biologischen Systemen der Fall ist. Das Emergenzmodell der Physik, das beschreibt, wie Ordnung aus Unordnung entsteht, ist eng mit den Computermodellen der Emergenz verwandt. Thermodynamische Emergenz versucht unter anderem, Phänomene des Zusammenspiels von Determinismus und Stochastik zu fassen. Der Prozess wird ebenfalls in eine mikroskopische und eine makroskopische Ebene unterteilt. Thermodynamische Prozesse können auf Mikroebene nur stochastisch beschrieben werden, auf Makroebene zeigen sie durchaus diskretes deterministisches Verhalten. In gewisser Weise ist dies eine Umkehrung der Prinzipien des Computers. Hier haben wir gerade auf mikroskopischer Ebene vollkommen deterministisches Verhalten, auf Makroebene dagegen werden zum Beispiel bei der Leistungsmodellierung von Computersystemen stochastische Modelle eingesetzt, um Vorhersagen über das Laufzeitverhalten von Prozessen zu treffen. Emergenz in Bezug auf ein Modell wird definiert als Abweichung des Verhaltens eines Systems vom Modell, das ein externer Beobachter sich von diesem System gemacht hat. Es ist klar, dass diese Form der Emergenz im Bereich der Programmierung nur durch Code-Reflexivitäten erreicht werden kann, ansonsten wäre das starre Programm immer das exakte Modell für das Verhalten des Systems.

Die bisher angesprochenen Beispiele können alle als ‘Mikroheterarchien’ gelten, bei denen in der Regel eine sehr große Zahl regelbasierter Einheiten miteinander in Wechselwirkung stehen. Die Regeln des Zusammenwirkens sind hierbei meist sehr einfach und es findet nur Interaktion mit den direkten Nachbarn statt. ‘Makroheterarchien’ zeichnen sich dagegen dadurch aus, dass die interagierenden Komponenten in sich eine komplexe Struktur besitzen können und meist nur relativ wenige Einheiten an der Interaktion beteiligt sind. Als Beispiel für eine Makroheterarchie kann die zwischenmenschliche Kommunikation genannt werden. Von echter Kommunikation kann man demzufolge nur sprechen, wenn der Ausgang der Kommunikation offen ist.

36 Siehe Zwischenspiel: Zelluläre Automaten & Selbstreproduktion.

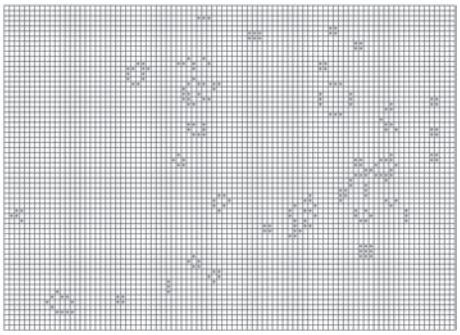


Abbildung 50/51:
Mikroheterarchie. *Game of Life* als regelmäßiges Raster gleichzeitig aktiver Einheiten. Zur Berechnung des Nachfolgezustandes wird jeweils Bezug auf die Zustände der direkten Nachbarn genommen (links). Makroheterarchie. Parallele Aktivitätsträger, die selbst bereits komplexe programmiersprachliche Objekte darstellen und die sowohl untereinander als auch mit dem Spieler interagieren (rechts).

Auf technischer Seite sind Computerspiele bzw. allgemein VR-Welten typische Vertreter. Bei technisch realisierten Heterarchien entstehen Interpretationsspielräume und Handlungsoffenheit durch das Ineinandergreifen von Abläufen, wobei die einzelnen Komponenten die Ziele der anderen in der Regel nicht kennen. Solche makroskopischen Heterarchien sind bisher wenig untersucht und werden meist im Trial-and-Error-Verfahren entwickelt.

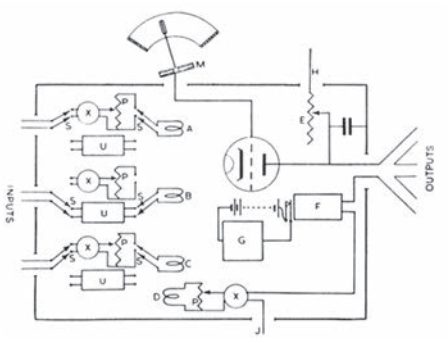


Abbildung 52/53:
W. R. Ashbys Homöostat aus vier Elektromagneten gilt als einer der ersten Versuche mit sich selbst organisierenden Systemen. Das analoge System kann gleichzeitig als früher Versuch gedeutet werden, Heterarchien systematisch zu untersuchen. Der Konstrukteur W. R. Ashby mit seinem Homöostaten (links). Schaltbild einer Einheit des Homöostaten (rechts).

1.4.5.3 Die Unschärfe des Interfaces

Das Bewusstsein, dass jede Rezeption eines Kunstwerks die aktive Partizipation des Betrachters erfordert, ist eine kritische Erkenntnis zeitgenössischer Ästhetik. Kein Kunstwerk kann demnach einem Betrachter genau das mitteilen, was der Künstler beabsichtigt. Marcel Duchamp nennt dies den 'persönlichen Kunst-Koeffizienten', der das Verhältnis zwischen dem 'Unausgedrückten-aber-Beabsichtigten' und dem 'Unabsichtlich-Ausgedrückten' beschreibt. In jeder ästhetischen Erfahrung kommt dem Betrachter selbst eine konstitutive Rolle zu. Auch das weiter oben angeführte Zitat Ecos zum offenen Kunstwerk ist in dieser Weise zu interpretieren. Wir können diese Einsicht aber auch auf Computerinterfaces und die hybride Maschine übertragen, indem wir fragen, was Interfaces leisten. In jedem Interface findet eine semantische Reduktion der komplexen Umgebung statt. Bei technisch aufwendigen Interfaces, die einen permanenten Datenstrom liefern – etwa einer Kamera – können aber im Prozess der Reduktion auch bereits wieder Nebeneffekte ins Spiel kommen. Das mit einer Kamera erzeugte Bild enthält niemals nur exakt die Informationen, die vom Kameramann intendiert waren.

Abbildung 54:
Doppelte Unschärfe. Die Ergebnisse eines (unscharfen) Analogrechners werden mit einer Kamera von einer Kathodenstrahlröhre abgefilmt.

Doch die Unschärfe des Interfaces beschränkt sich nicht auf die eingangsseitigen Meßprozesse, sondern gilt in gleichem Maße für die Ausgabeprozesse. Der Rechenprozess selbst läuft auf Digitalrechnern zwar vollkommen deterministisch ab, bei der Umsetzung numerischer Ergebnisse und symbolischer Repräsentationen in Bilder werden aber ebenfalls unvermeidlich Nebeneffekte generiert. Das tatsächliche Aussehen eines computergenerierten Bildes kann nicht im Detail vom Programmierer vorhergesehen werden. Er wird bei einiger Erfahrung mit seinem Programm bestimmte fixe Qualitäten der Bilder erkennen können, das einzelne Bild wird aber ebenfalls Details und Interpretationsspielräume enthalten, die jenseits der bewussten Setzung durch den Programmierer liegen. Durch die Umformung der Zeichen und der numerischen Werte in den Ausgabe-Interfaces kommt also ebenfalls Unbeabsichtigtes ins Spiel und es ergeben sich neue semantische Potentiale. Winograd und Flores bezeichnen diese Phänomene auch als 'unbeabsichtigte Repräsentation'.[37] So können zum

Beispiel sehr regelmäßige und dynamische Muster entstehen, wenn Inhalte von Datenstrukturen grafisch dargestellt werden. Beispielsweise nimmt der Betrachter Kreise auf dem Bildschirm wahr, obwohl weder ein explizites Konstruktionsprinzip für Kreise im Code zu finden ist, noch der Programmierer in irgendeiner Phase der Programmentwicklung an Kreise gedacht hat. Festzuhalten ist: Interfaces können eingabeseitig semantische Potentiale reduzieren und sowohl eingabe- als auch ausgabeseitig neue semantische Potentiale generieren. Die Echtzeit-Interaktion mit einem Computer auf der Basis von Bildern, Tönen, Schmerzempfindungen, etc. löst unvermeidlich sowohl physiologisch als auch kulturell determinierte Wahrnehmungsprozesse aus und ist damit anders zu beschreiben als das Input-/Output-Verhalten einer zeichenbasierten Turingmaschine.

1.4.6 Abschließendes

„Man kann sagen, wir genießen nicht die Erziehung, die notwendig ist für unser Dasein in dieser Welt. Wir entbehren des technischen Bildungsbegriffs, der die technische Ausbildung umsetzt in einen ethisch sinnvollen Charakter dieser technischen Welt. Denn jede Bildung ist ja Bildung für die bewohnte Welt und damit Anteil am Sein dieser Welt, das ästhetisch, ethisch, religiös und intellektuell gerechtfertigt sein will. Einerseits müsste diese Erziehung ja jene funktionale Bildung vermitteln, die uns – fachmännisch – den Ort der Welt anweist und ausfüllen lässt. Andererseits aber ist es notwendig, dass die funktionale Bildung ergänzt wird durch eine andere, die uns die geistige Schöpfungsfähigkeit, die intellektuelle, rationale Kraft der Hervorbringung der Gedanken, Erkenntnisse, Ideen, Aktionen ästhetischer, ethischer, religiöser und technischer Art erhält. Die gesamte Pädagogik muss der technischen Welt, der technischen Existenz und ihrer zweifachen Tendenz – auf den technischen Fachmann einerseits und die geistige Existenz andererseits – angepasst werden.“

Max Bense[38]

Im Unterschied zum klassisch-mechanischen Maschinenbegriff, der sich eng an der Zweckbestimmung der einzelnen Maschine orientierte, rückt heute das Verhalten ins Zentrum der Maschine. Diese Verschiebung gegenüber dem klassischen Begriff erfolgte nicht zuletzt vor dem Hintergrund der Computerentwicklung, einem zweifelsfrei mechanischen System, das sich aber neben materiellen auch durch immaterielle Bestandteile auszeichnet. Die programmierbare Maschine besteht genaugenommen aus zwei universellen Maschinen: 1. Der universellen Hardware, die zur physikalischen Welt gehört und deren Gesetzen gehorchen muss (Naturgesetze, Energieverbrauch, Alterungsprozesse, Defekte, etc.). In diese Hardware wird das konkrete Verhalten per Programm eingeschrieben. Damit die Maschine sich diskret und deterministisch verhält, muss die unscharfe physikalische Realität ‘domestiziert’ werden. In Bezug auf die Zustände der digitalen Maschine gelingt die Domestizierung, bei den

37 Vgl. Winograd und Flores, in: Erkenntnis Computer Verstehen, S. 155.

38 In: Technische Existenz, in: ders. Ausgewählte Schriften Band 3, Ästhetik und Texttheorie, S. 129.

in den Interfaces wirksamen Messprozessen kann sie dagegen aus prinzipiellen Gründen nicht gelingen. 2. Universelle Programmiersprachen erlauben es, algorithmische Verhaltensweisen auf der Basis von Zeichenfolgen zu beschreiben. Die so beschriebenen Software-Maschinen unterliegen nicht mehr den Grenzen der Physik, sondern nur noch den Grenzen unseres Denkens. Die einzige Begrenzung ist unsere Vorstellungskraft, die durch die Struktur der Programmiersprache ihren Rahmen erhält. Diese in der Logik verwurzelte Maschine besitzt die ganze Strenge und Schärfe, die formale algorithmische Systeme auszeichnen: Endlichkeit, Elementarität und Determiniertheit.

Wir hatten festgestellt, dass typische Programmstrukturen, allge-meine Organisationsstrukturen und Psychostrukturen Sedimentationen sozialer Beziehungen sind. Diese Beziehungen sind allgemeine historisch-gesellschaftliche Strukturprinzipien, die sich durch ständig wiederholte Interaktionen als Grundmuster ablagern und sich in Maschinen- und Organisationsstrukturen vergegenständlichen (Sequenzen, bedingte Wiederholungen, Abstraktionen, Module, Hierarchien und Heterarchien). Mit ihrer formalen Beschreibung stehen sie auch als Konstrukte in Programmiersprachen zur Verfügung und sind der Simulation durch die Maschine zugänglich. Die symbolische Maschine wurde soweit vorangetrieben, dass nicht nur Daten und Anweisungen durch die gleichen Zeichen beschrieben werden. Die Stärke der Maschine erwächst gerade daraus, dass von der jeweiligen Situation abhängt, welche Zeichen als Daten und welche als Anweisungen betrachtet werden. Da die Bedeutung der Zeichen interpretationsabhängig ist, können sie nicht nur zwischen Operator und Operand changieren, sondern auch Objekte auf unterschiedlichen Abstraktionsniveaus beschreiben. In der universellen programmierbaren Maschine ist damit ein vorläufiger Höhepunkt der semiotischen Abstraktion erreicht und die Kulturgeschichte kann (nach Flusser) ihre Umkehrung antreten. Umkehrung heißt einerseits, dass durch hinreichend präzise Simulation von Ausschnitten der Realität Handlungsalternativen getestet werden können und aufgrund der Ergebnisse in die Realität eingegriffen werden kann. Das Gebiet der Computersimulation verfolgt genau diese Eingriffe durch Antizipation. Sie erlaubt, Neues auszuprobieren und experimentell zu erforschen und durch Variation subjektive wie objektive Konsequenzen von Verhaltensalternativen in der Simulation zu testen, um die lukrativste Alternative für die reale Handlung auszuwählen. Wichtiger erscheint aber der zweite Aspekt, nämlich dass mit dem Eindringen des Computers in alle Lebensbereiche eine eigene technische Realität geschaffen wird, die nichts mehr repräsentiert, sondern die selbst Wirklichkeit konstruiert. Der Mensch nimmt aktiv und kreativ am Prozess der Wahrnehmung und der Erkenntnis teil. Der Geist ist nicht der Spiegel einer objektiven Außenwelt, es gibt keine Eigenschaften, die der Interpretation vorausgehen. Das menschliche Wissen wird vielmehr durch Zeichen und Symbole bestimmt, die historisch, gesellschaftlich und kulturell geprägt werden. Wenn es stimmt, dass die dem Menschen eigene Fähigkeit

zur Bildung von Begriffen und Symbolen ein fundamentales und notwendiges Element für das Verständnis und die Konstruktion von Wirklichkeit ist, dann ist zweifellos der Computer ein interessantes Werkzeug, um diese Prozesse zu begleiten und zu unterstützen. Insbesondere im Zusammenspiel von entäußertem Handlungsdenken, das durch Codes beschrieben und von Prozessoren ausgeführt wird, mit komplexen Interfaces, die Handlungen in Wahrnehmungen von Bildern, Klängen, Bewegungen, etc. umsetzen, entfaltet der Computer sein imaginatives Potenzial. Durch die Schnittstellen tritt der zeichenhafte Charakter der Maschine für den Nutzer wieder in den Hintergrund, er wird gewissermaßen durch das Interface dissimuliert. Für den Programmierer erschließt sich aber gerade durch die Kombination der symbolisch beschreibbaren Handlungsprozesse mit den wahrnehmungsorientierten Interfaces ein experimenteller Raum, der für die Entwicklung, Erprobung und endgültige Realisierung von konzeptuellen Überlegungen genutzt werden kann.

Als eine der großen Irrlehren unserer Zeit kann die nachhaltige Pflege des Zerrbildes von den zwei Kulturen – der naturwissenschaftlich/technischen auf der einen und der kulturell/geistigen auf der anderen Seite – angesehen werden. Die Techniker/Wissenschaftler können angeblich nicht verstehen, welchen Zwecken sie dienen und die Geisteswissenschaftler sind nicht in der Lage zu erfassen, was sie kritisieren. Vor dem Hintergrund der massiven Eingriffe, die nicht nur Informations- und Kommunikationstechnologien, sondern gegenwärtige Technologien ganz allgemein auf unsere Lebensführung ausüben, wird diese Trennung zunehmend riskant. Gleichzeitig bleibt auf diese Weise das eigentliche Potenzial und die Faszination der Denktraditionen, die sich in der Struktur der programmierbaren Maschine niedergeschlagen haben, weiten Teilen der Geisteswissenschaften entzogen. Es geht im vorliegenden Buch keinesfalls darum, einer „technoid verklärenden oder verklärten Medientheorie“[39] das Wort zu reden oder computerbasierte Medien als „durch die Eigenlogik der Artefakte, unabhängig von Kultur und Lebensform“ bestimmte Erscheinungen zu betrachten. Vielmehr unterstützen wir die Forderung einer „historischen Anthropologie der Medien“. „‘Anthropologie’ beschreibt die Tatsache, dass Menschen ihre Natur im Medium der Artefakte entwickeln. ‘Anthropologie der Medien’ bedeutet, dass die Herstellung des ‘Wesens Mensch’ wesentlich in und durch Medien geschieht, was nicht nur für die primären Ordnungen des Entwurfs, sondern auch für deren Nachbedenken gilt.“[40] Wir wollen uns abschließend vorbehaltlos der Forderung Max Benses anschließen, dass die gesamte Bildung gleichzeitig auf die technischen Kenntnisse einerseits und die geistige Existenz andererseits abzielen muss. In diesem Sinne versuchen wir, die bloße Sachbeherrschung mit dem geistigen Sachverstand für das Gebiet der Computerprogrammierung zusammen zu bringen.

39 Hans Ulrich Reck, „Inszenierte Imagination“ – Zu Programmatik und Perspektiven einer „historischen Anthropologie der Medien“, S. 232.

40 Ebd., S. 243.