Zur Genese des informatischen Programmbegriffs: Begriffsbildung, methaphorische Prozesse, Leitbilder und professionelle Kulturen

Hellige, Hans Dieter

Veröffentlichungsversion / Published Version
Arbeitspapier / working paper

Forschungszentrum Nachhaltigkeit

Hans Dieter Hellige

Zur des Genese des informatischen Programmbegriffs: Begriffsbildung, metaphorische Prozesse, Leitbilder und professionelle Kulturen

artec-paper Nr. 108
Dezember 2003

ISSN 1613-4907

artec - Forschungszentrum Nachhaltigkeit
Enrique-Schmidt-Str. 7
Postfach 330 440
28334 Bremen
http://www.artec.uni-bremen.de

Das Forschungszentrum Nachhaltigkeit ist eine Zentrale Wissenschaftliche Einrichtung der Universität Bremen. Es wurde 1989 zunächst als Forschungszentrum Arbeit und Technik (artec) gegründet. Seit Mitte der 90er Jahre werden Umweltprobleme und Umweltnormen in die artec-Forschung integriert. Das Forschungszentrum bündelt heute ein multi-disziplinäres Spektrum von – vorwiegend sozialwissenschaftlichen – Kompetenzen auf dem Gebiet der Nachhaltigkeitsforschung. „artec“ wird nach wie vor als ein Teil der Institutsbezeichnung beibehalten.

Das Forschungszentrum Nachhaltigkeit gibt in seiner Schriftenreihe „artec-paper“ in loser Folge Aufsätze und Vorträge von MitarbeiterInnen sowie ausgewählte Arbeitspapiere und Berichte von durchgeführten Forschungsprojekten heraus.

Impressum

Herausgeber:

Universität Bremen

artec Forschungszentrum Nachhaltigkeit
Postfach 33 04 40
28334 Bremen
Tel.: 0421 218 61800
Fax.: 0421 218 98 61800
Web: www.uni-bremen.de/artec

Kontakt:

Andrea Meier
E-Mail: andrea.meier@artec.uni-bremen.de

Zur des Genese des informatischen Programmbegriffs: Begriffsbildung, metaphorische Prozesse, Leitbilder und professionelle Kulturen

Hans Dieter Hellige

> Bei unseren Tätigkeiten, gleich welcher Art sie seien und in welcher Situation wir sie vollbringen, begleiten uns Interessen, Absichten, Hoffnungen, Wünsche, Vorstellungen, Erwartungen, Maximen, Regeln – kurz, ein ganzes Bündel von motivierenden und orientierenden Verfaßtheiten. Wir lassen uns von ihnen nicht nur begleiten, sondern auch leiten. Meist wirken solche Orientierungen beiläufig und wie selbstverständlich, gar mit einer gewissen Zwangsläufigkeit. Sie können einander auch widersprechen, oder: unmittelbares Ziel unserer Tätigkeit und übergeordnetes Leitbild mögen nicht zusammenpassen. *Frieder Nake 2003, S. 344*

1. Die hermeneutische Anlyse von Metaphern, Leitbildern und professionellen Kulturen in der Technikgenese

Die Informatik und hier speziell die Software-Entwicklung, Software–Ergonomie und die Gesellschaftstheorie der Informatik haben seit den 70er Jahren mit der Entdeckung der Bedeutung von Mentalen Modellen, Benutzer-Modellen und Metaphern sowie mit dem Perspektiven- und Model-Power-Konzept das Wissen über hermeneutische Prozesse zwischen Technikproduzenten und -nutzern stark erweitert. Der besondere Konstruktionsgegenstand Software hat hier Erkenntnisse zu Tage gefördert, die der Mechanik- bzw. Elektrokonstruktionslehre verschlossen blieben. Die Techniksoziologie und die Technikgeschichte widmen sich seit den 80er Jahren intensiver der Analyse von Leitbildern, Metaphern und generell kultureller Aspekte der Technikgenese. Schließlich beteiligten sich auch Psychologie und Philosophie verstärkt an der Erforschung von Modellbildungs- und Übertragungsprozessen in der Technik. Von einer elaborierten Hermeneutik des technischen Gestaltens kann man trotz dieser Forschungsanstrengungen aber noch immer nicht sprechen. Dazu fehlt es noch an einer Zusammenschau der verschiedenen Phänomene und erst recht an einer Systematisierung hermeneutischer Prozesse im Technischen Handeln.

In einer groben Gliederung lassen sich allgemein-gesellschaftliche Horizonte wie Technische Kulturen, Technikstile und Technikbilder von den besonderen Erfahrungs- und Vorverständnis-Horizonten der Technikentwickler unterscheiden (vgl. Hellige 1995a, S. 21 ff.). Die allgemein-gesellschaftlichen wie die besonderen technisch-wissenschaftlichen Horizonte sind aufgrund des Hintergrundcharakters von Vorverständnissen nur partiell und dies auch nur durch spezifische hermeneutische Methoden und

Konzepte rational explizierbar. Zu ihnen gehört auf der einen Seite das vor allem in der Informatik entwickelte Perspektivenkonzept, das durch die Gegenüberstellung unterschiedlicher Sichtweisen verabsolutierte Standpunkte auflöst. Auf der anderen Seite stehen eine Reihe bereichs- oder aspektspezifischer Orientierungsmuster wie Mentale Modelle, Metaphern, Leitbilder und Konstruktionsstile. Deren Zusammenspiel soll im Folgenden am Beispiel metaphorischer Prozesse in der Genese der Informatik untersucht werden.

Metaphern bilden, bezogen auf den gesellschaftlichen Gehalt, den Übergang von mentalen Modellen zu den Leitbildern. Denn auch hierbei wird an vertraute Gestaltmuster und Erfahrungen angeknüpft, um möglichst übergangsgerechte Lösungsmuster zu generieren. Metaphern sind keinesfalls nur eine Begleiterscheinung der Modellierung von Arbeits- und Handlungsabläufen auf dem Rechner, wie es die informatische Metaphernforschung vielfach nahelegt. Die Historie der Gestaltfindung bei Telegrafen, Telefonen, elektrischen Herden, Waschmaschinen sowie von mechanischen Rechen- und Schreibmaschinen zeigt vielmehr, daß man Metaphern offenbar als einen wesentlichen Bestandteil der Artefaktkonstruktion ansehen muß. Vor allem bei der Mensch-Maschine-Schnittstelle scheint der Rückgriff auf vertraute Lösungsmuster unverzichtbar, sei es, weil neue Gestaltmuster hier besonders schwer zu schaffen oder den Benutzern zu vermitteln sind.

Dabei läßt sich zeigen, daß Metaphern in Entwicklungsprozessen nicht nur als kognitive Medien kreativen Kombinierens zu betrachten sind, wozu Thomas P. Hughes (1991, S. 83 ff.) und die technikgenetische Metaphern-Studie von Mambrey, Paetau und Tepper (1995) neigen. Modell- und Gestalt-Übertragungen sind im hohen Maße auch un- oder halbbewußte Momente des Vorverständnisses, also hermeneutischer Natur. Neben der spielerisch-bewußten Konstruktion mit Metaphern gibt es die Vorfixierung auf bekannte Muster und Sichtweisen. Dadurch wird der Lösungsraum u. U. von vornherein eingeengt. Man sollte deshalb immer die Ambivalenz von Metaphern im Blick behalten: die Vorprägung durch Bestehendes und die kreative Neuschöpfung. Aus dem hermeneutischen Charakter von Übertragungsprozessen ergibt sich m. E. eine veränderte Sicht der Rolle der Metaphernforschung in der Technikbewertung. Sie sind weder bloße Geistesblitze, die man nur konstatieren, aber nicht beeinflussen kann. Sie sind aber auch nicht der Angelpunkt der Erklärung und Bewertung von Technikgeneseprozessen. Metaphern geben Auskunft über Vorverständnisse, Absichten, Benutzerbilder usw., sagen aber nichts über die systemische Problemstruktur und die Langzeitdynamik einer Technik aus.

2. Zur Bedeutung metaphorischer Übertragungsprozesse in der Wissenschaftsgenese der Informatik

Ein großer Teil informatischer Begriffe ist aus metaphorischen Übertragungen aus anderen Technikbereichen oder Wissenschaften hervorgegangen. Das Metaphernsortiment der Informatik ist sogar, dies haben Richard Lynch, Peter Mambrey, Michael Paetau, August Tepper und Carsten Busch betont, besonders bunt gemischt. Deren Erforschung hat sich bisher vor allem an geisteswissenschaftliche Methoden angelehnt und dabei die Metaphern aus dem Kontext gelöst und wie literarische Metaphern interpretiert. Dies mag bei bildhaften oder unmittelbar einsichtigen Gestaltmetaphern wie Baum, Stapel, Schleife, Sprung und Bug, Virus usw. angemessen sein, nicht jedoch bei komplexeren Analogiebildungen wie Programm, Programmiersprache, Schichtenmodell oder Architektur. Diese transportieren über Gestaltanalogien hinaus ganz spezifische professionelle Sichtweisen. Sie haben oft implizit oder explizit Leitbildfunktion. Dies wird besonders bei konkurrierenden Metaphern deutlich: Software Engineering hat sich ab 1968 gegen die um 1965 noch protegierte Software-Architektur durchgesetzt, während das Computer Engineering ab 1970 ganz eindeutig von der Computer-Architektur verdrängt wurde. Zur Erklärung reichen da Etymologien und literarisch-philosophische Metapherndeutungen nicht mehr aus, hier muß der Zusammenhang von Metaphern und Leitbildkomplexen in professionellen Kulturen diskursanalytisch betrachtet werden. Am Beispiel der Entstehung des Programmbegriffes möchte ich nun zeigen, welche Bedeutung metaphorische Prozesse bei der Entstehung einer neuen Disziplin haben können und wie Denkweisen bestehender professioneller Kulturen über sie auf die neu entstehende einwirken.

Obwohl die Begriffe Programm, Programmiersprachen zu den Kernbegriffen der Informatik zählen, ist deren genauere Entstehung noch weitgehend unerforscht. Die meisten Autoren historischer Rückblicke arbeiten ohnehin mit einem universalen Programmbegriff: dieser wird für die Tempeltore des Heron von Alexandria, die mittelalterlichen Uhrwerke, die frühneuzeitlichen Automaten und Spielwerke ebenso verwendet wie für moderne Computer. Durch die Allgegenwart des Programmbegriffs wird jedoch verdeckt, mit welcher Begrifflichkeit man jeweils arbeitet und warum man Jahrhunderte lang ohne einen von den jeweiligen materiellen Speichermedien Walze, Lochkarten oder Stecktafeln losgelöste abstrahie-rende Bezeichnung ausgekommen ist. Denn für Babbage, Ludgate, Torres und Vannevar Bush stehen die Ketten der

„Operational Cards“ bzw die Lochstreifen noch genauso für das prozedurale Programm selber wie die Lochkarten und die Walze für Jacquard und die Erbauer mechanischer Musikinstrumente.

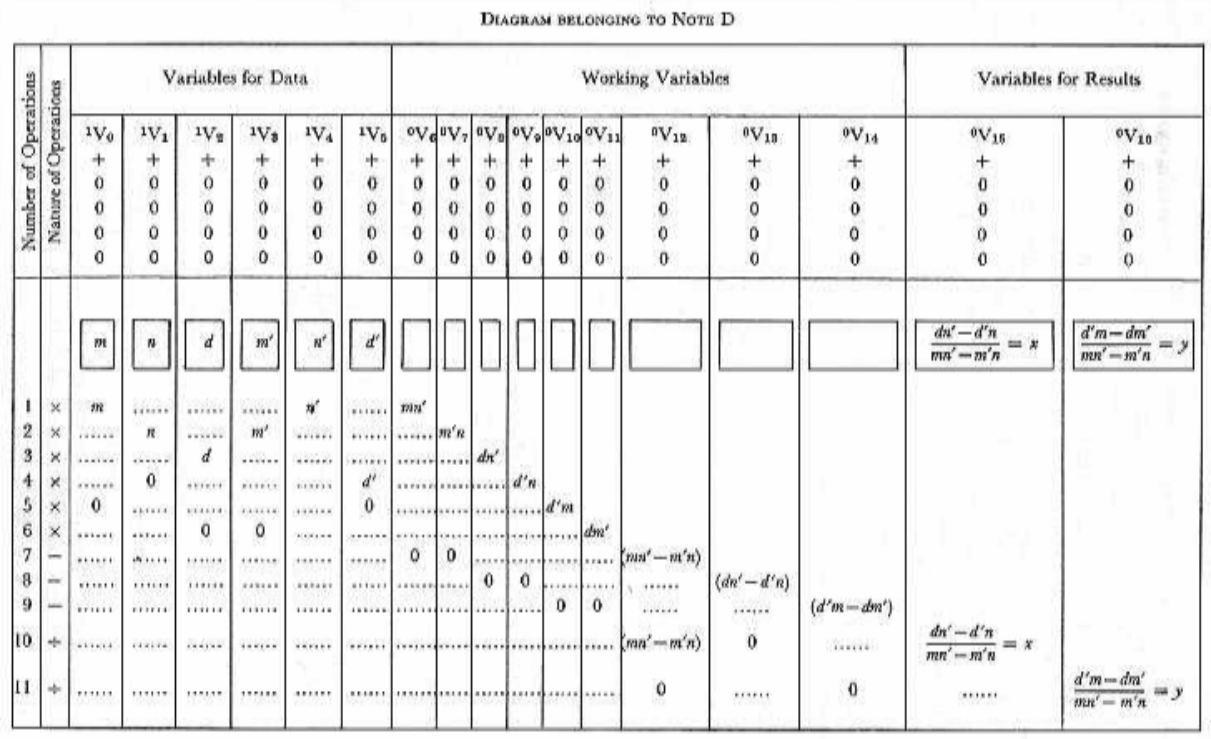


DIAGRAM BELONGING TO NOTE D

Number of Operations	Nature of Operations	Variables for Data						Working Variables									Variables for Results	
		$^1V_0$	$^1V_1$	$^1V_2$	$^1V_3$	$^1V_4$	$^1V_5$	$^0V_6$	$^0V_7$	$^0V_8$	$^0V_9$	$^0V_{10}$	$^0V_{11}$	$^0V_{12}$	$^0V_{13}$	$^0V_{14}$	$^0V_{15}$	$^0V_{16}$
		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		m	n	d	m'	n'	d'										$\frac{dn'-d'n}{mn'-m'n}=x$	$\frac{d'm-dm'}{mn'-m'n}=y$
1	×	m	.....	.....	.....	n'	.....	mn'										
2	×	.....	n	.....	m'	.....	.....	.....	$m'n$									
3	×	.....	.....	d	.....	.....	.....	.....	.....	dn'								
4	×	.....	0	.....	.....	.....	d'	.....	.....	.....	$d'n$							
5	×	0	.....	.....	.....	.....	0	.....	.....	.....	.....	$d'm$						
6	×	.....	.....	0	0	.....	.....	.....	.....	.....	.....	.....	dm'					
7	−	.....	.....	.....	.....	.....	.....	0	0	.....	.....	.....	.....	$(mn'-m'n)$				
8	−	.....	.....	.....	.....	.....	.....	.....	.....	0	0	.....	.....	.....	$(dn'-d'n)$			
9	−	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	0	0	.....	.....	$(d'm-dm')$		
10	+	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	$(mn'-m'n)$	0	.....	$\frac{dn'-d'n}{mn'-m'n}=x$	
11	+	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	0	.....	0	.....	$\frac{d'm-dm'}{mn'-m'n}=y$

Diagramm des Programmablaufes mit Jacquard-Lochkarten (Ada Lovelace in: Menabrea, Note D)

So bezeichnet Babbage, dem „mettre en carte“ bei Jacquard entsprechend, Programmiervorgänge meistens als „arrangement“ oder „combination“ der verschiedenen „sets of cards“ zu einer „order by means of cards“ bzw. „chain of operational cards“ (Babbage, bes. S. 45 f.; Merrifield, S. 57) Daneben taucht aber bereits die Übersetzungsmetapher auf: „In this light the cards are merely a translation of algebraical formulae, or, to express it better, another form of analytical notation.“ (Menabrea) Schließlich bedienen sich Babbage und Ada Lovelace der viel zitierten Analogien zu den Jacquardwebstühlen, indem sie davon sprechen, mit dem „system of cards“ algebraische Muster zu weben: „We may say most aptly, that the Analytical Engine weaves algebraical patterns just as the Jacquard-loom weaves flowers and leaves.“ (Ada Byron-King, Note A in Menabrea) Percy E. Ludgate wandelt zwar mit seinen Lochstreifen den Ablauf der Bedien- und Rechenprozesse ab, bleibt aber ansonsten ganz der Babbage-Terminologie verpflichtet und greift einmal sogar auf die Ada-Metaphorik zurück (Ludgate 1909). Leonardo Torres y Quevedo lehnt sich einerseits an den Babbage-Begriff des Arrangements von Operationen an, doch entwickelt er

andrerseits schon ein anthropomorphes Verständnis des ‚Programmablaufes', wonach das „automaton" wie ein intelligentes Wesen bestimmten Regeln folgt und bei jedem Schritt während des Fortschreitens der Operationen Entscheidungen trifft (Torres 1914). Vannevar Bush schließlich bleibt in seinen Vorkriegsschriften bei der „sequence of operations" und der „chain of actions", die jeweils durch Impulse angestoßen werden, während er bei seinem Lochkarten-gesteuerten Differential Analyzer von 1942 von einer „task" des „ set up" spricht (Bush 1940, S. 343; Bromley, S. 185).

Wie und wann der Loslösungsprozeß beginnt und welche Rolle die Programmmetapher dabei spielt, ist noch ziemlich im Dunkeln. Als frühesten Beleg für den Programmbegriff im Computerbereich gilt nach David Alan Grier John Mauchlys berühmtes Memorandum von 1942 „Use of High Speed Vacuum Tube Devices for Calculating", das die ENIAC Genese einleitete. Es gibt zwar bereits Manuskripte von Konrad Zuse, in denen der Programmbegriff angeblich schon 1938 vorkommt, doch diese Datierungen sind wenig glaubwürdig, da in diesen Dokumenten bereits von Programmier*sprachen*, algorithmischen Sprachen und vom bedingten Sprung die Rede ist. So viel hat Zuse nun doch noch nicht vorweggenommen (vgl. Hellige 2003, S.).

2. Programmbegriffe in der professionellen Kultur der Steuerungs- und Regelungstechnik

2.1 Von De Leeuws „program machine" zum „program control" und Programmregler

An dem Mauchly-Beleg von 1942 ist keinem bisher aufgefallen, daß hier nur von einem „programming" bzw. „program *device*" die Rede ist. Es ist damit eine Art Programmschaltung oder Stecktafel gemeint, die in dem geplanten System verketteter Rechenmaschinen den Ablauf zwischen den verschiedenen Rechenwerken steuert: „this program device is capable of arranging a cycle of different transfers and operations of this nature with perhaps fifteen or twenty operations in each cycle" (Mauchly 1942, S.330). Ein solcher Mechanismus ähnelt den damals in der Steuer- und Regelungstechnik üblichen „selbsttätigen Arbeitsfolge- und Zeitgebereinrichtungen". Für die „bekannteste Klasse" von Regelungen bzw. Steuerungen, die „nach einem vorgeschriebenen Plan verändert wurden" hatte sich in Deutschland spätestens seit dem Ende der 30er Jahre der Begriff „Programmregler" eingebürgert, der seinerseits auf den älteren Begriff „program control" in USA zurückging (Schmid 1941; Engel, Oldenbourg 1944, S. 200 f.).

Es handelt sich dabei um eine Automatisierungstechnik an der Nahtstelle zwischen Steuerungs- und Regelungstechnik, bei der das Programm durch Zeit- und Wertestellknöpfe oder Steckverbindungen eingestellt und zum Teil sogar in auswechselbaren Sichtscheiben angezeigt wurde: „[...]der Arbeiter wird durch den Programmregler bei der Arbeitsabwicklung unterstützt, so daß er seine Aufmerksamkeit anderen wichtigeren Dingen zuwenden kann. Der Programmregler besitzt eine den Arbeitsschritten entsprechende Anzahl von Sichtscheiben, die nacheinander, entsprechend den vorher gewählten Zeitabschnitten aufleuchten [...] Durch Auswechslung der Sichtscheibe und Neueinstellung der Zeitstellknöpfe läßt sich der Programmregler leicht auf eine andere Arbeitsfolge umstellen." Für Werkzeugmaschinen waren noch komplexere Programmregler vorgesehen, die neben der Zeit und den Arbeitsschritten auch noch die Geschwindigkeit und die Werkzeugwahl steuerten (Schmid, 1941, S. 69). Kompliziertere Steuerungsabläufe wurden entweder wie im Fall von Stromerzeugungs- oder -verteilungsanlagen über motorisch angetriebene „Steuerwalzen" oder wie beim Satz und Druck nach dem Vorbild der Jacquard-Webstühle durch Lochkarten gesteuert: „Der Lochkarte wird die Denk- und Willensfunktion des Menschen unmittelbar übertragen. Die Anordnung der Löcher bezeichnet die Symbole für die Betätigungsvor-

gänge der Maschine.“ (Meiners, S. 86 ff.; Strauch 1937, S. 476) Für die Planung und Erstellung der Schalt- bzw. Steueranordnung entstand bei der AEG Mitte der 30er Jahre bereits das Modellierungsinstrument des „Schaltfolgendiagramms“, das die einzelnen Schaltschritte mit den jeweiligen Verriegelungs- und Schaltbedingungen in „Relaissymbolik“ graphisch darstellte. Es war so einfach und nah am Produktionswissen des Anlagenpersonals, daß es noch Ende der 60er Jahre vom Erfinder der Speicherprogrammierbaren Steuerung Richard Morley als Vorbild der SPS-Programmiermethode des „Kontaktplans“ („ladder diagram“) gewählt wurde (Meiners 1936, S. 35-42; Scharf 1989, S. 9; Kröck 1991).

Die Idee des „program control“ entstand bereits Jahrzehnte zuvor in den USA, spätestens 1920/22. In diesen Jahren entwickelte nämlich der Werkzeugmaschinen- und Automatisierungsexperte Adolph Lodewyk De Leeuw (geb. 1861), Consulting Editor am „American Machinist“, ein seinerzeit Aufsehen erregendes, später aber wieder in Vergessenheit geratenes Konzept für die automatische Steuerung von Maschinen. Nach einer ersten Ideenskizze in der Zeitschrift „Industrial Management im Juni 1920 gab er 1922 in einer auch als Buch erschienenen Artikelserie im „American Machinist“ einen Gesamtüberblick über die „Methods of Machine Tool Design“ und die Zukunftssichten von „automatic machines“ speziell für die Kleinserienfertigung. Um das Auslastungsproblem besonders von Werkzeugmaschinen zu lösen, entwarf er ein „system of control of automatic functions of machine elements“, in dem alle „automatic machining operations“ der sich abwechselnden Bohr-, Dreh-, Schraubwerkzeuge usw. zu einer Art Bearbeitungszentrum integriert waren. Die einzelnen Bearbeitungsgänge waren nicht direkt verkettet, sie wurden vielmehr nach einem zu vor festgelegten Plan aufgerufen und stoppten nach dem Bearbeitungszyklus von selber, um dem nächsten „predetermined cycle“ das Feld zu überlassen.

Die Steuerung erfolgt dabei durch ein „auxiliary mechanism .[...] which we will call the ›program‹“, nämlich „an endless chain which is advanced one link every time a cycle comes to an end“: „I would call this style a ›program machine‹“ Diese „program mechanism“ genannte Vorrichtung. sollte im halbautomatischen Betrieb dem „operator“ durch einen Buchstaben-Code den nächsten Arbeitsschritt anzeigen und im vollautomatischen Betrieb alle „instructions“ [sic!] als Kette von Schaltfolgen abwickeln. Mit diesem „program mechanism“ löste sich die konzipierte Maschinensteuerung von

den üblichen Kopierautomaten und Fühlersteuerungen, die noch *analoge* Einzweckautomat waren, es entstand bereits die Idee einer *codebasierten* Mehrzwecksteuerung für unterschiedlichste Maschinen und Prozessfolgen. Das auslösende Moment für den Wechsel vom mechanisch fixierten Programmablauf zur programmierbaren Code- bzw. Lochstreifensteuerung bildete eine Metapher. So heißt es bei De Leeuw: „Instead of a chain, a perforated roll of paper might be used, very much like the music rolls for a player piano." Für die komplizierten Verknüpfungen einer ganzen Reihe von Werkzeugen mit einfachen, parallelen und wiederholten Bearbeitungszyklen entwickelte er sogar schon eine Art Assemblersprache, so daß es bei ihm. bereits drei verschiedene Notationen gab: einzelne Buchstaben oder ganze Worte bei der Programmplanung und für die Anzeigen sowie Lochmuster für die Maschine. Allerdings hatte er noch keine Symbol- und Formelsprache im Sinn, wie sie Franz Reuleaux schon 1875 für Getriebeelemente entwickelt und für eine codebasierte Maschinenkonstruktion konzipiert hatte (Reuleaux 1875, S. 243-71).

Mit dem Programm- und Befehlsbegriff und der Codesteuerung ging De Leeuws „program machine" von 1920-22 über die bekannten programmgesteuerten Maschinenkomplexe der 20er-30er Jahre hinaus. Denn während diese die Verknüpfung *direkt* in Lochstreifen, Lochkartenstapeln, Stecktafeln oder in der Verdrahtung fixierten, dachte er bereits an ein flexibles Arrangement von Abläufen, bei dem allein über Umcodierung neue Programme generiert und variiert werden konnten. Für diesen neuen Aktionsraum der Kombination und Variation von „instructions" jenseits der reinen Lochstreifen-Codierung benötigte De Leeuw einen Begriff und er fand ihn im Ensemble von Musikprogramm und Lochbandsteuerung bei den automatischen Klavieren. Es hat den Anschein, dass die Metapher der lochbandgesteuerten automatischen Klaviere nicht nur wie seinerzeit die Jacquard-Steuerung für Babbage die Konstruktionsidee geleitet, sondern auch den Anstoß für die Verwendung des Programmbegriffs geliefert hat. Jedenfalls leitet sich der informationstechnische Programmbegriff damit ursprünglich nicht aus dem Radioprogramm ab, sondern aus der

2.1 Leitbild „program control" bei Wallace Eckert, John P. Eckert und Mauchly: Automatisierung durch Integration der Programmierschritte in „program devices"

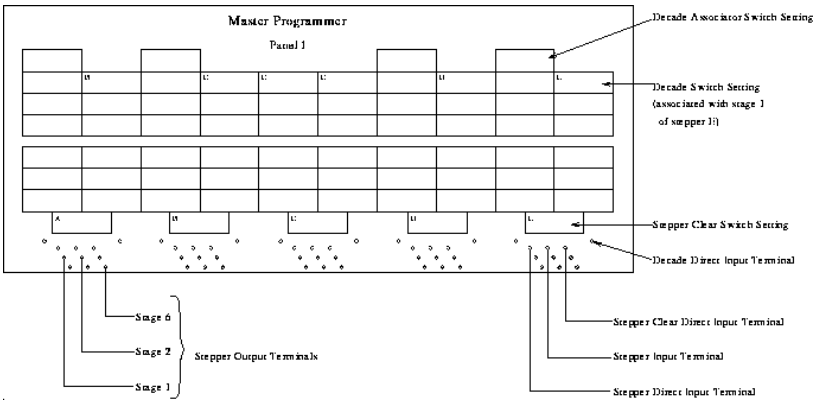
Mit der Einführung der Begriffe „program" und „instruction" war konzeptionell im Bereich der Maschinensteuerungen die Trennung des Programms von seinen materiellen Trägern und den manuellen Verknüpfungen eingeleitet. Über diese Metaphern entwickelte sich in den 20er –30er Jahren die steuerungs- und regelungstechnische Konstruktionstradition des „program control". Ihr Leitbild war nicht mehr die virtuose Handhabung der manuellen Verknüpfungen wie bei den Vertretern des „manuellen Programmierens" (vgl. Hellige 1998), sondern eine auf Automatisierung zielende Zusammenführung der Programmierschritte in „program devices". Um 1933 tauchte dieses Konzept und der Programmbegriff erstmals im Bereich der Rechen- und Lochkartenmaschinen auf: Wallace J. Eckert verband für astronomische Berechnungen verschiedene Lochkartenmaschinen zu einem Maschinenkomplex, der durch ein „device called a control switch" gesteuert wurde (W. Eckert 1940; Ceruzzi 1998, S 18). Dessen „program" steuerte eine zentrale Ablaufsteuerung für bis zu 20 Arbeitsschritte, die er „mechanical programmer" nannte, also eine anthropomorphe Metapher, die sich an die Bezeichnung für Programmacher im Radio- Konzert- und Theaterbetrieb anlehnte (zur Radiotradition des Programmbegriffs siehe Coy. 1998)

Das neue Programmierungskonzept prägte auch noch ganz eindeutig die ENIAC-Genese. Auch dort handelte es sich, wie gesagt, um eine aufgabenspezifisch verbindungsprogrammierte Maschinenverkettung, deren einzelne Arbeitsschritte mit Hilfe von „program devices" konfiguriert und gesteuert wurden. Aus der Bindung an diese professionelle Kultur erklärt sich auch das auffällig kontrolltechnische Verständnis des Programmierens bei den beiden aus der Elektrotechnik kommenden ENIAC-Designern John Mauchly und John Presper Eckert. In den ENIAC-Proposals und –Berichten von 1942-44 taucht der Programmbegriff nämlich fast ausschließlich in Verbindung mit „devices", „circuits", „pulses" oder „switches" auf, die durch eine „program control unit" eingestellt und gesteuert werden. So heißt es etwa zur ENIAC-Programmsteuerung im Proposal vom April 1943: „A unit which contains the necessary control units for initiating the various steps of the calculation in their proper order. The program control unit can be equipped, if desired, which a punch-card programm selector to facilitate rapid set-up of different problems."(Mauchly, Eckert, Brainerd 1943, zit. nach Burks, Burks, S. 335 f.) Auch in den Berichten, Memoranden und

Vorträgen der Jahre 1946/46, so in dem von Adele Goldstine verfaßten ENIAC-Report vom 1. Juni 1946, der ersten öffentlichen Darstellung der Anlage durch beide Goldstines im gleichen Jahr, den Moore-Lectures von Eckert und Mauchly sowie in Mauchlys Skizze der EDVAC-Programmierung von 1947 bezieht sich der Programmbegriff fast nur auf die Hardware-gebundene Programmabwicklung, während die Programmiertätigkeiten mit „planning“ und „preparation of problems“ bezeichnet werden. Selbst hinter dem „master programmer“, der 1944 zu der bis dahin dezentralen Kontrollstruktur hinzukam, verbirgt sich nicht etwa der Chefprogrammierer, sondern wie bei Wallace Eckerts „mechanical programmer“ ein „central control switchboard“. An ihm wurden durch Steckverbindungen die lokalen „controls“ der Unterprogramme zu einem „single program“ sequenziert (A. Goldstine, 1946). Begrifflich fiel dabei das abstrakt-logische Programm noch immer mit der Gesamtheit der Kontrollgeräte und technischen Steuerungsprozesse zusammen.

Die Bindung an das steuerungs- und regelungstechnische Programmierkonzept erhielt bei den ENIAC-Entwicklern eine folgenreiche kognitive Lenkungsfunktion. Sie bewirkte, daß sie die Komplexität der Programmierung unterschätzten und von einer schnellen Automatisierung der Programmierprozesse ausgingen. Bereits 1944 formulierte Eckert in dem berühmten Memorandum „Disclosure of Magnetic Calculating Machines“ (Kopie 1.2.1945, Ms. 29.1.1944), das die wohl früheste Formulierung des ‚Programmspeicher’-Konzepts enthält, auch das Leitziel des „automatic programming“:

ENIAC : MASTER PROGRAMMER



Wenn „discs“ oder „drums“ und „multiple shaft systems“ zur Anwendung kämen, „a great increase in the available facilities and for allowing automatic programming of the facilities and processes involved may be made, since longer time scale are provided. This greatly extends the usefulness and attractiveness of such a machine. This programming may be of the temporary type set up on alloy discs or of the permanent type on etched discs.“ (Eckert 1944) Bei dem Moore-Lectures propagierte Mauchly eine

möglichst weitgehende Automatisierung der verschiedenen „steps“ von der „preparation of problems“ bis zu „set up“ und „operation“, um so die „costs of computing“ zu senken: „If these steps can be systematized and reduced to more or less routine operations, there is hope of performing them automatically.“ (Mauchly 1946, S.33 f.) „Automatic programming“ entwickelte sich sehr bald zu einem Schlagwort und Leitbild, das die Computer Community lange Zeit fehlleitete, weil es die Komplexität der Programmierung vergessen ließ. Es ist daher kein Zufall, daß die unterschiedlichen logischen und arbeitsorganisatorischen Prozesse der Programmerstellung nicht im Umfeld der professionellen Kultur des „program control“ und der in ihrer Tradition stehenden ENIAC-Entwickler herausgearbeitet wurden, sondern bei Mathematikern wie John von Neumann und Alan Turing, die sowohl praktisch wie theoretisch in die Rechner- und Programmkonstruktion involviert waren.

3 Die Genese komplexer informatischer Programmbegriffe in der professionellen Kultur der Mathematik und der frühen Theorie des Computing

3.1 ‚Programmieren' als konstruktives Planen und Problemlösen bei v. Neumann

John v. Neumann entwickelte mit Hermann H. Goldstine und Arthur Burks zusammen 1945-47 ein Sechsstufenmodell der Programmiervorgänge, wobei klar zwischen den nicht automatisierbaren konzeptionellen, konstruktiven und dynamisch-analytischen Aufgaben einerseits und eher routinisierbaren statischen Codierungsprozessen andrerseits unterschieden wurde. Mit Blick auf die Mannigfaltigkeit der Programmiertätigkeiten verzichtete v. Neumann, der ansonsten mit einer ganzen Reihe von Metaphern aus Natur- und Technikwissenschaften experimentierte, hier auf technische, biologische oder anthropomorphe Analogiebildungen. Die anfangs noch verwendete Übersetzungsmetapher nahmen er und Goldstine 1947 explizit zurück: „Since coding is not a static process of translation, but rather the technique of providing a dynamic background to control the automatic evolution of a meaning, it has to be viewed as a logical problem and one that represents a new branch of formal logics." [1] Es findet sich nicht einmal ein einheitlicher Begriff für die Gesamtheit der Programmiervorgänge, ja von Neumann weigerte sich beharrlich in allen von ihm allein verfassten Texten, mit dem Programmbegriff zu arbeiten, und zwar auch noch nach 1948, als in der Community die Bezeichnung „von Neumann's programming method" aufkam (Clippinger 1948). Aber auch in gemeinsamen Berichten mit Burks und Goldstine sind „Planning" und „Coding" die Leitbegriffe, während „Programming" nur selten erscheint.[2] Und wenn, dann auch nur in der partiellen Bedeutung der Bildung von Funktionen aus im Rechner gespeicherten Subroutinen für Addier-, Gleitkomma- und dergleichen Operationen: „these operations can be programmed by means of others." (Burks, Goldstine, v.Neumann 1946, S. 70). Mit den Stufen Planung und logische Problemlösung als übergeordneten Tätigkeiten und der Formulierung des Codes als ausführende Detailarbeiten wird die hierarchische Arbeitsteilung als Hintergrundperspektive erkennbar, die selber wiederum in eine Gesamtsicht der am Computing beteiligten Akteursgruppen eingebettet ist. Und diese implizite soziale Metaphorik ließ sich offenbar nicht mit dem damals noch stark kontrolltechnischen Verständnis des Programmbegriffs verein-baren.

[1] Goldstine, v.Neumann 1946, S 30 („translate the problem (once it is logically reformulatet and made explicite in all its details) into the code."; Goldstine, v.Neumann 1947, S. Coding deckt hier noch alle Operationen ab!

[2] Siehe hierzu die Schriften über Computerdesign und Automatentheorie in: von Neumann 1963, bes. Burks, Goldstine, v. Neumann 1946; Goldstine, v. Neumann 1947-48; vgl. allgemein Goldstine 1972 und Aspray 1990

Von Neumanns Planungsbegriff für Programmierungsaufgaben korrespondiert auffällig mit seinem Organisationsbegriff für das Rechnerdesign. Diesen hatte er zusätzlich zu seinem Organmodell der Rechnerstrukturen und dem Neuronenmodell der dynamischen Rechenprozesse in die frühe Architectural Community eingebracht (vgl. Hellige 2003, S. 418-22). So wie er mit dem Organisationsbegriff auf die Bewältigung von Ziel- und Designkonflikten sowie auf Ressourcen- und Dimensionierungsprobleme in der Computerkonstruktion aufmerksam machen wollte, so sollte auch der Planungsbegriff auf die Aufgabenkomplexität und Vielfalt des Tätigkeitsspektrums hinweisen. Soziale Metaphorik und Multiperspektivität erklären sich so nicht zuletzt aus von Neumanns dezidierter „stakeholder-view" des Computing, die man bisher weitgehend übersehen hat: er betrachtete nämlich die Bauprinzipien von „computing machines" sowohl aus dem Blickwinkel der konstruierenden Ingenieure, der Algorithmen entwerfenden Mathematiker als auch der „user" – bei ihm gab es 1946 bereits die Redewendung „the user desires" (Goldstine, v. Neumann 1946, S. 22) – Die „user" faßte er idealtypisch unter der Bezeichnung „logician" zusammen: „a hypothetical person or group of persons really fitted to plan scientific tools" (ebda., S. 1) Obwohl durch und durch Mathematiker, reflektierte von Neumann Computer- und Programmstrukturen als komplexes, Zielkonflikt-behaftetes Konstruktionsproblem, für das ihm die Planungs- und Organisationsmetapher adäquater erschien. Eine sehr ähnliche Designauffassung veranlaßte um 1960 Frederick Brooks, den Computer Engineering-Begriff durch die Architekturmetapher zu ersetzen (Hellige 2003, S.436-48).

3.2 Programmieren als Überbrücken von Sprachdifferenzen zwischen Mensch und Maschine bei Turing

Ein Übergang von technisch-naturwissenschaftlichen zu sozialen Metaphern findet sich auch bei Alan Turing. Durch seinen Einstieg in die Debatte über theoretische Fragen der Berechenbarkeit und der Strukturen „universaler Maschinen" hatte er bereits 1945/46 die Programmierung im Spannungsfeld zwischen menschlicher und maschineller Intelligenz angesiedelt und wesentlich als ein Problem der Sprachdifferenz gedeutet. Die Lücke zwischen der symbolischen Sprache der Maschinen und der Alltagssprache des Menschen schuf ein Kommunikationsproblem: „It schould be possible to describe to the operator in ordinary language within the space of an ordinary novel. These instructions will be not quite the same as the instructions which are normally given to a

computer, and which give him credit for intelligence.“ (Turing 1946, S. 39) Das Nebeneinander verschiedener Notationen und „languages“ war nur über eine Kette von „translations“ und exakte Sprachen zu überwinden: „The language in which one communicates with these machines, i.e. the language of instruction tables, form a sort of symbolic logic. The machine interprets whatever it is told in a quite definite manner without any sense of humor or sense of proportion [...]“ (Turing 1947, S. 122). Über die Sprach- und Übersetzungsmetapher erschloß sich auch ihm die ganze Vielfalt und Komplexität der ‚Kommunikationsprozesse’ mit dem Rechner. Doch im Gegensatz zu von Neumann griff er 1947 den Programmbegriff der ENIAC-Gruppe auf. Er löste ihn jedoch von dem engen Hardware- und Kontrolltechnik-Bezug und gab ihm aus seiner anthropomorphen Perspektive die Bedeutung eines Bündels von Tätigkeiten mit unterschiedlichen Intelligenzanforderungen und Sprachvermögen.

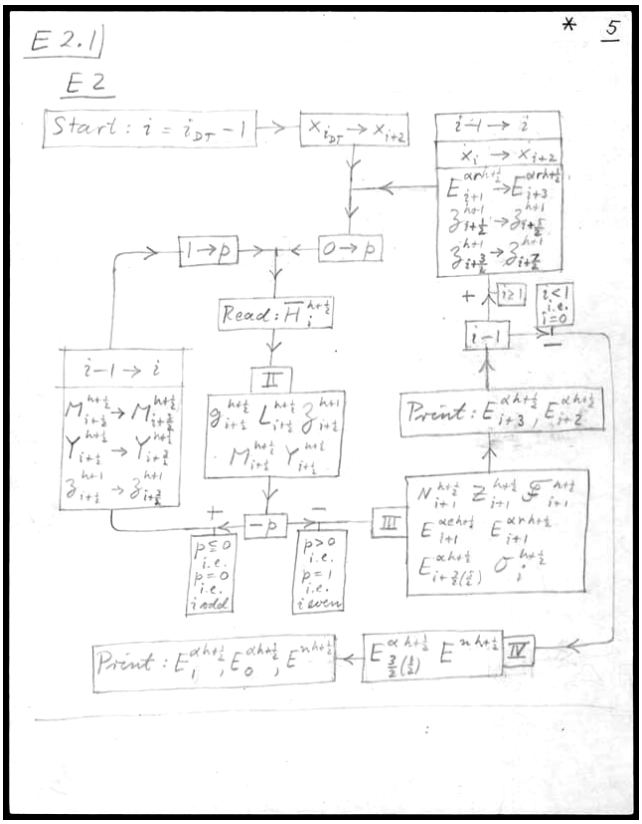
Die Prozesse der Programmerstellung und –durchführung sah er dabei bereits wie lange zuvor schon Babbage und eher implizit John von Neumann als ein hierarchisch strukturiertes System der Arbeitsteilung. An der Spitze stand als „master“ der „programmer“, unter dem er nicht mehr ein „device“ verstand, sondern den ‚human programmer’. Die Metapher war damit wieder zum programmierenden Subjekt zurückgekehrt. Durch seine zugleich personale und soziale Sicht der Programmierung wurde Turing offenbar der Schöpfer des Begriffs „Programmierer“ im Computerbereich. Unter dem „programmer“ sorgten die „librarians“ für die Ordnung und Pflege der Programme, Routinen und Subroutinen, während die „girls“ die Werte und Daten erfaßten sowie die „servants“, die den Rechner mit Lochkarten fütterten. Das Rechenzentrum wurde so nahezu ein Abbild der akademischen Arbeitsteilung mit Bibliothek, Sekretariat und Hilfskräften. Den weniger Qualifizierten drohte er bereits mit der baldigen Verdrängung durch Fortschritte der Computertechnik. Die Maschinen waren zwar in dem hierarchisch organisierten Intelligenzverbund zwar die Sklaven“: „It is also true that the intention in constructing these machines in the first instance is to treat them as slaves, giving them only those jobs which have been thought out in detail“ Turing 1947, S. 122) Doch sie waren auch gelehrige Schüler („pupils“), die von ihren „masters“ lernten, wenn man ihnen nur genügend Speicherplatz und Entfaltungsspielraum gewährte. Wörtlich heißt es in der berühmten Passage: „What we want is a machine that can learn from experience. The possibility of letting the machine alter its own instructions provide the mechanism for this, but this of course does not get us very

far.“ (S. 123) Turings Überlegungen von 1945 über die Sprachdifferenzen zwischen Mensch und Maschine mündeten so zwei Jahre später in der „idea of a machine with intelligence“, die selber im System der Arbeitsteilung als Konkurrent auftrat. Es ist auffällig, dass George Stibitz zur gleichen Zeit in ähnlicher Weise die Vorgänge in einem Rechner als hierarchische Kette kooperierender Schichten darstellte, als eine „series of levels of intelligence“ (Stibitz 1948, S. 96 ff.; vgl. Hellige 2003, S. 423 ff.) . Hier kündigte sich bereits der Übergang von den stärker technischen und naturwissenschaftlichen Metaphern und Modellvorstellungen der Anfangszeit zu soziomorphen Analogien und Modellierungen der entwickelten Computer- und Programmiertechnik an. Doch die eigentliche Zeit der Hierarchie-, Pyramiden- und Schichtenmodelle, der Produktionslinien und Fabriken kam erst mit den 60er und 70er Jahren.

Mithilfe der Sprach- und Übersetzungsmetapher hatte sich Turing die Verschiedenheit des menschlichen und maschinellen Sprachbaus (Pflüger 1993) erschlossen und von daher die sozialen Prozesse der Programmierung in den Blick bekommen. Er befreite damit die Begriffe „program“ und „programmer“ von der Hardware-Fixierung in der professionellen Kultur des „program control“. Er entwickelte wohl als erster ein Sprachkonzept der Programmierung, allerdings noch ohne den Begriff der Programmiersprache. Dessen erstes Erscheinen ist wie das der Begriffe Betriebssysteme und Software noch immer nicht ermittelt. Mit dem Leitbild der „intelligenten Maschine“ überfrachtete Turing zugleich das erweiterte Programmverständnis mit anthropomorphen Ansprüchen, die die Technik auf absehbare Zeit nicht einlösen konnte. So setzte Turing dem unterkomplexen kontrolltechnischen Programmverständnis ein hyperkomplexes entgegen, das sich am Ende ebenfalls als eine fehlleitende Simplifikation erweisen sollte.

4 Die Anfänge von Rationaliserungsmetaphern in der Programmierung und das Problem von metaphorischen Prozessen mit unreflektierter Leitbildfunktion

John v. Neumann hat beiden Tendenzen widerstanden, dem Glauben an eine schnelle Realisierung des „automatic programming“ wie an eine baldige Überwindung der Sprachdifferenzen zwischen Mensch und Maschine. Er bewahrte sich aus seiner gleichermaßen theoretisch-mathematischen und nutzerbezogenen Problemsicht den Blick für die Verschiedenheit der Prozesse als auch für den dynamisch-konstruktiven Charakter des Programmierens. Dieses war für ihn gerade nicht „a mere question of translation (of a mathematical text into a code) but rather a question of providing a control scheme for a highly dynamical process, all parts of which may undergo repeated and relevant changes in the course of this process.“



Flowdiagram von John v. Neumann

Das adäquate Modellierungsinstrument für diese dynamischen Beziehungen war für die von-Neumann-Gruppe das graphische „Flußdiagramm“, das traditionell in den Ingenieurwissenschaften zur Darstellung von komplexen Anlagenstrukturen sowie von Stoff- und Energieflüssen verwendet wurde. Diese *„flowdiagrams*“ wurden erst im Laufe der 50er und frühen 60er Jahre von den *„flowcharts“* abgelöst, die sich im Namen und in der Symbolik an die Flowcharts der beiden Gilbreth anlehnten. Dieser

Wandel steht aber bereits im Zusammenhang mit dem breiten Einzug sozialer Metaphern und soziomorpher Modellbildungen in diesem Zeitraum. Es ging jetzt nicht mehr vorrangig um die Darstellung und Planung komplexer Strukturen, sondern um eine Rationalisierung der Softwareproduktion. Dies wird nicht zuletzt daran deutlich, dass man auch für die Strukturierung des Softwareentwicklungsprozesses ab 1962/63 ebenfalls auf ein tayloristisches Rationaliserungsinstrumente wie das Gannt-Diagramm zurückgriff. Daraus entwickelten sich später die Phasenmodelle des Software Engineering, insbesondere das bekannte Wasserfallmodell von Barry Boehm. Das Vorbild der industriellen Fertigungsmethoden für die Programmentwicklung wurde hier kaum mehr hinterfragt. Die Modellimporte erhielten auf diese Weise eine weitgehend unreflektierte Leitbildfunktion. Nachdem aber erstmal das Bewußtsein verloren gegan-gen ist, daß es sich um konstruierte Analogiebildungen mit begrenzter Reichweite handelt, kann es leicht zu metaphorischen Zirkelschlüssen kommen. So bereits bei einer der frühesten Gleichsetzung der Programmierung mit der industriellen Fertigungsplanung bei Alwin Walther.

Walther, der in der Tradition verketteter Maschinensysteme 1942-44 mit Hilfe von steckbaren "Kopplungstafeln" übliche elektromechanische Rechenwerke, Tabulatoren bzw. Lochkartengeräte zu programmgesteuerten Rechenautomaten verband, interpretierte bereits 1946, 1952 und 1955 in Vorträgen den "Aufbau von Rechenautomaten" als Abbild und zugleich als Muster bzw. Kernstück der vollautomatischen Fabrik. Die Kopplungstafel fungiert danach als "Befehls-Steuerwerk", das die "Einzelwerke" steuert, bzw. als "Gehirn", das "Arbeitsbefehle" an die Gliedmaßen und Muskeln" gibt, d.h. die Addier- und Multiplizierwerke (Walther 1956, S. 17). Letztere verglich er auch mit dem Fließband, während das "Kommandowerk" das "Analogon zur Betriebsleitung" darstelle (ebda., S. 38). Auch das noch so mühsame und langwierige Programmieren unterwarf er der ‚Fabrikordnung': die einzelnen Prozesse und Phasen wurden exakt der Fabrikationsplanung von nachgebildet.

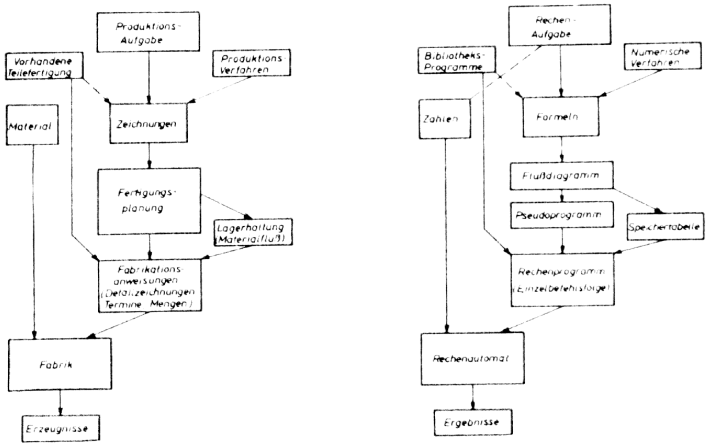


Bild 15

Vergleich zwischen der Planung einer Fabrikation und dem Programmieren eines Rechenablaufs

Die gleichzeitige Modellierung der Programmfertigung als Abbild der industriellen Fertigungsplanung und die Deklarierung des Rechners und seiner Programmstruktur als Vorbild für die Fabrikorganisation lassen Ursprungs- und Zielbereich der Metaphernbildung verschwimmen. Wenn aber die soziomorphe Modellierung von Technostrukturen und die technomorphe Modellierung von Sozialstrukturen sich gegenseitig verfestigen, dann erhalten sie undurchschaubaren Ideologiecharakter.

Eine mit historischen Vergleichen arbeitende hermeneutische Bewertung aktueller Techniken vermag zwar selber keine direkte Aussagen über falsche oder richtige Übertragungsvorgänge zu liefern. Sie kann aber typische Problem- und Fehlerkonstellationen aufführen. Dazu gehören:

- die Fixierung auf bestimmte Metaphern als Folge professioneller Kulturen
- eine zu direkte Musterübertragung aus der alten in die neue Technik
- die mögliche Vererbung alter Probleme und impliziter Grenzen in die neue Technik
- metaphorische Zirkelschlüsse mit unreflektierter Leitbildfunktion

Da diese Problem- und Fehlerkonstellationen in der Informatik nicht selten auftreten und da aus inadäquaten Übertragungen oft folgenreiche Irrwege entstehen können, hat eine historisch-vergleichende Metaphernforschung im Rahmen einer Technikhermeneutik wichtigen Aufgaben zu leisten. Denn die Vorurteilstruktur des Verstehens ist nicht, wie Gadamers Hermeneutik-Auffassung es nahe legt, unausweichlich, die Reflexion kann vielmehr, wie Habermas es ihm entgegenhielt, das „Medium der Überlieferung“ grundlegend wandeln: „Die transparent gemachte Vorurteilsstruktur kann nicht mehr in der Art eines Vorurteils fungieren. Die Kraft der Reflexion vermag den Anspruch der Tradition auch abzuweisen.“

5 Literatur

Aspray, William, John von Neumann, and the Origins of Modern Computing, Cambridge, MA, London 1990

Babbage, Charles, On the Mathematical Powers of the Calculating Engine (1837), in: Randell, Brian (Hrsg.), The Origins of Digital Computers. Selected Papers, Berlin, Heidelberg, New York 1973, S. 17-52

Barnes, Ralph M., Motion and Time Study, 4. Aufl. New York 1958

Bromley, Alan G., Analog Computing Devices, in: Aspray, William (Hrsg.), Computing before Computers, Ames, IA 1990, S. 156-199

Burks, Arthur W., From ENIAC to the Stored-Program Computer: Two Revolutions in Computers, in: Metropolis, Nicholas C.; Howlett, Jack; Rota, Gian-Carlo (Hrsg.), A History of Computing in the Twentieth Century. A Collection of Essays, New York, London 1980, S. 311-344

Burks, Arthur W.; Burks, Alice R., The ENIAC: First General-Purpose Electronic Computer, in: Annals of the History of Computing, 3 (1981) 4, S. 310-399

Burks, Arthur W., Goldstine, Herman H.; Neumann, John von, Prelimininary Discussion of the Logical Design of an Electronic Computing Instrument, Part I, Volume 1, Institute for Advanced Study Princeton, N. J. Juni 1946; 2. Aufl. 1947; wiedergedruckt in: Taub, Abraham H. (Hrsg.), Complete Works of John von Neumann, 6 Bde.Oxford (UK), New York 1961-63, Bd. 5, S. 34-79

Busch, Carsten, Metaphern in der Informatik. Modellbildung, Formalisierung, Anwendung, Wiesbaden 1998

Bush, Vannevar, Arithmetical Machine (Ms., 2. März 1940); gedruckt in: Randell, Brian (Hrsg.), The Origins of Digital Computers. Selected Papers, 3. Aufl. Berlin, Heidelberg, New York 1982, S. 337-344

Ceruzzi, Paul E., A History of Modern Computing, Cambridge, MA 1998

Clippinger, Richard F., A Logical Coding System Applied to the ENIAC (Electronic Numerical Integrator and Computer) Ballistic Research Laboratories Report No. 673 Project No. TB3-0007 of the Research and Development Division, Ordnance Department 29 September 1948 Aberdeen Proving Ground, Maryland; als Internet-Dokument unter: http://ftp.arl.army.mil/~mike/comphist/48eniac-coding/

Coy, Wolfgang, Hat das Internet ein Programm? Vortrag auf dem medienwissenschaftlichen Symposium der Universität Konstanz am 30.10.98, Internet-Version: http://waste.informatik.hu-berlin.de/Coy/Coy_Internet_11-98.html

De Leeuw, Adolph Lodewyk, Methods of Machine Tool Design, in: American Machinist (1922) Heft 57/17, Oktober 1922, S. 639-642 (Schlußabschnitt: “The Program Machine and Its Future possibilities”)

De Leeuw, Metal Cutting Tools, their principles, action and construction, New York 1922

Eckert, John Presper, A Preview of a Digital Computing Machine, Lecture 10, in: Campbell-Kelly, Martin; Williams, Michael R. (Hrsg.), The Moore School Lectures. Theory and Techniques for Design of Electronic Digital Computers (Charles Babbage Institute (Hrsg), Reprint Series for the History of Computing, Bd. 9), London, Los Angeles, San Francisco 1985, S. 109-126

Eckert, John Presper; Mauchly, John W., Automatic High-Speed Computing: A Progress Report on the EDVAC. Report No. W-670-ORD-4926, Supplement No. 4, Moore School Library, University of Pennsylvania, Philadelphia, 30. September 1945, University of Pennsylvania, Philadelphia 1946

Eckert, John Presper; Mauchly, John W.; Goldstine, Herman H.; Brainerd, John Grist, Description on the ENIAC, Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia Nov. 1945

Eckert, Wallace J., Punched Card Methods in Scientific Computation, Thomas J. Watson Astronomical Computing Bureau, New York 1940

Engel, F. V. A.; Oldenbourg, Rudolf C., Mittelbare Regler und Regelanlagen. Grundlagen, Aufbau und Anwendung, Berlin 1944

Gilbreth, Frank B.; Gilbreth, Lilian M., Process Charts, in: Transactions of the ASME 43 (1921), Paper 1818, S. 1029-1050

(Goldstine, Adele K.), Report on the ENIAC (Electronic Numerical Integrator and Computer), Technical Report 1, 2. Bde., Philadelphia, 1.Juni 1946; als Internet-Dokument unter: http://ftp.arl.army.mil/~mike/comphist/46eniac-report

Goldstine, Herman H., A Report on the ENIAC, Part I, Bd. I und II, Technical Description on the ENIAC, Moore School of Electrical Engineering, University of Pennsylvania , Philadelphia 1946

Goldstine, Herman H., The Computer from Pascal to von Neumann, Princeton, N. J. 1972

Goldstine, Herman H.; Goldstine, Adele, The Electronic Numerical Integrator and Computer (ENIAC), in: Mathematical Tables and other Aids to Computation 2 (1946) Juli, S. 97-110; wiedergedruckt in: Randell, Brian (Hrsg.), The Origins of Digital Computers. Selected Papers, Berlin, Heidelberg, New York 1973, S. 333-347; 2. Aufl. 1975; 3. Aufl. 1982, S. 359-373; ebenso in: Annals of the History of Computing 18 (1996) 1, S. 10-16

Goldstine, Herman H.; Neumann, John von, Planning and Coding of Problems for an Electronic Computing Instrument, 3 Bde. Institute for Advanced Study, Princeton 1947-1948; wiedergedruckt in: Taub, A. H. (Hrsg.), Complete Works of John von Neumann, 6 Bde.Oxford (UK), New York 1961-63, Bd. 5, Part II, Volume 1:S. 80-151; Volume 2: S. 152-214; Volume 3: S. 215-235; und in: Aspray, William; Burks, Arthur W. (Hrsg.), Papers of John von Neumann on Computing and Computing Theory, Cambridge/ Mass., London, Los Angeles, San Francisco 1987, Volume 1:S. 151-222; Volume 2: S. 223-285; Volume 3: S. 286-306

Goldstine, Herman Heine; Neumann, John von, On the Principles of Large Scale Computing Machines (Ms. 1946), gedruckt in: Taub, Abraham H. (Hrsg.), Complete Works of John von Neumann, 6 Bde.Oxford (UK), New York 1961-63, Bd. 5, S. 1-34

Grier, David Alan, The ENIAC, the Verb "to program" and the Emergence of Digital Computers, in: Annals of the History of Computing 18 (1996) 1, S. 51-55

Habermas, Jürgen, Zur Logik der Sozialwissenschaften

Hellige, Hans Dieter (1993), Von der programmatischen zur empirischen Technikgeneseforschung: Ein technikhistorisches Analyseinstrumentarium für die prospektive Technikbewertung, in: Technikgeschichte, Bd. 60, Nr. 3, S.186-223

Hellige, Hans Dieter, Der 'begreifbare' Rechner: Manuelles Programmieren in den Anfängen des Human-Computer Interface, in: Ingrid Rügge, Bernd Robben, Eva Hornecker, Willi Bruns, (Hrsg.), Arbeiten und Begreifen: Neue Mensch-Maschine-Schnittstellen, Münster, Hamburg 1998, S. 187-200

Hellige, Hans Dieter, Technikleitbilder als Analyse-, Bewertungs- und Steuerungsinstrumente: Eine Bestandsaufnahme aus informatik- und computerhistorischer Sicht, in: ders. (Hrsg.), Technikleitbilder auf dem Prüfstand. Das Leitbild-Assessment aus Sicht der Informatik- und Computergeschichte, Berlin 1996, S. 13-36

Hellige, Hans Dieter, Leitbilder im Time-Sharing-Lebenszyklus: Vom "Multi-Access zur "Interactive On-line Community", in: ders. (Hrsg.), Technikleitbilder auf dem Prüfstand. Das Leitbild-Assessment aus Sicht der Informatik- und Computergeschichte, Berlin 1996, S. 205-234

Hellige, Hans Dieter, Die Genese von Wissenschaftskonzepten der Computerarchitektur: Vom „system of organs“ zum Schichtenmodell des Designraums, in: ders. (Hrsg.), Geschichten der Informatik. Visionen, Paradigmen und Leitmotive, Berlin, Heidelberg, New York 2003, S. 411-470

Hughes, Thomas P., Die Erfindung Amerikas. Der technologische Aufstieg der USA seit 1870, Müchen 1991

Jüttemann, Herbert, Mechanische Musikinstrumente. Einführung in Technik und Geschichte, Frankfurt a. M. 1987 (H mus 96.5/757)

Knuth, Donald E., Von Neumann's First Computer Program, in: Computing Surveys 2 (1970) Dezember, S. 247-260

Kröck, Alwin, SPS,der Schlüssel zur Automa-tisierung, in : Technische Rundschau, 47/1991,S.68-73.

Leonhard, Adolf, die selbsttätige Regelung, Berlin, Göttingen, Heidelberg 1962 (1. Auflage 1940)

Leonhard, Adolf, Eine Systematik und zweckmäßige Einteilung der verschiedenen Regelungsarten, in: Elektrotechnik und Maschinenbau 58 (1940) 51/52, S. 541-546

Ludgate, Percy E., On a Proposed Analytical Machine, in: Scientific Proceedings of the Royal Dublin Society 12 (1909) 9, S. 77-91; wiedergedruckt in: Randell, Brian (Hrsg.), The Origins of Digital Computers. Selected Papers, Berlin, Heidelberg, New York 1973, S. 71-85

Lynch, Richard K., On Analytical ‘Engines’, Data ‘Architectures’ and Software ‘Engineers’: Metaphoric Aspects of the Development of Computer Terminology." Ph.D. Diss., Columbia University Teachers College, 1993

Mambrey, Peter; Paetau, Michael; Tepper, August, (1994), Technikentwicklung durch Leitbilder. Neue Steuerungs- und Bewertungsinstrumente, Frankfurt a. M., New York 1995

Mauchly, John W., The Use of High Speed Vacuum Tube Devices for Calculating (Aug. 1942), wiedergedruckt in: Randell, Brian (Hrsg.), The Origins of Digital Computers. Selected Papers, Berlin, Heidelberg, New York 1973, S.329-332

Mauchly, John W., Digital and Analogy Computing Machines, Lecture 1, in: Campbell-Kelly, Martin; Williams, Michael R. (Hrsg.), The Moore School Lectures. Theory and Techniques for Design of Electronic Digital Computers (Charles Babbage Institute (Hrsg), Reprint Series for the History of Computing, Bd. 9), London, Los Angeles, San Francisco 1985, S. 25-40

Mauchly, John W., Preparation of Problems for EDVAC-Type Machines, in: Proceedings of a Symposium on Large Scale Digital Calculating Machinery, Jan. 1947, wiedergedruckt in: Randell, Brian (Hrsg.), The Origins of Digital Computers. Selected Papers, Berlin, Heidelberg, New York 1973, S. 365-369

Mauchly, John W.; Eckert, John Presper; Brainerd, John Grist, Report on an Electronic Diff. Analyzer, Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, 12. April 1943

Menabrea, Luigi F., Sketch of the Analytical Engine Invented by Charles Babbage, Esq., Bibliothèque Universelle de Geniève, No. 82, Okt. 1842, translated into English with editorial notes by the translator, translated into English with editorial notes by the translator, Augusta Ada, Countess of Lovelace, in: Taylor’s Scientific Memoirs, Bd. III, Okt. 1843, Art.29., S. 666-731 wiedergedruckt in:

Morrison, Philip; Morrison, Emely (Hrsg.), Charles Babbage and His Calculating Engines. Selected Papers by Charles Babbage and Others, New York 1961; Netzversion: http://psychclassics.yorku.ca/Lovelace/menabrea.htm

Merrifield, C. W., Report of the Committee .. appointed to consider the avisability and to estimate the expense of constructing Mr. Babbage's Analytical Machine, and of printing tables by its means, Report of the British Association for the Advancement of Science, Dublin 1878, S. 92-102, London 1879, wiedergedruckt in: Randell, Brian (Hrsg.), The Origins of Digital Computers. Selected Papers, Berlin, Heidelberg, New York 1973, S. 53-63

Nake, Frieder, The Display as a Looking-Glass. Zu Ivan E. Sutherlands früher Vision der grafischen Datenverarbeitung, in: Hellige, Hans Dieter, (Hrsg.), Geschichten der Informatik. Visionen, Paradigmen und Leitmotive, Berlin, Heidelberg, New York 2003, S. 339-365

Neumann, John von, First Draft of a Report on the EDVAC, Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia 30. Juni 1945; wiedergedruckte korrigierte Fassung nach dem Originalmanuskript, hrsg. von Michael D. Godfrey in: Annals of the History of Computing 15 (1993) 4, S. 27-67

Neumann, John von (1963), Design of Computers, Theory of Automata and Numerical Analysis, in: ders. :Collected Works, 1903-1957, hrsg. von Taub, A. H., 6 Bde. Oxford (UK) 1961-63, Bd V

Oldenbourg, Rudolf C.; Sartorius, Hans, Dynamik selbsttätiger Regelungen, München Berlin 1944

Oxford English Dictionary, 2. Auflage Band XII, Oxford 1989, Artikel: program, programme, programmer, programming S. 589-592

Pflüger, Jörg-Martin (1993), Über die Verschiedenheit des maschinellen Sprachbaues, in: N. Bolz, F. Kittler, Chr. Tholen (Hrsg.) Computer als Medium, München, S. 161-181

Pflüger, Jörg-Martin (2002), Language in Computing, in: Doerries, Matthias (Hrsg.): Experimenting in Tongues: Studies in Science and Language, Stanford University Press, 2002, S.

Reuleaux, Franz, Lehrbuch der Kinematik, 1. Band: Theoretische Kinematik. Grundzüge einer Theorie des Maschinenwesens, Braunschweig 1875

Richter, Siegfried, Wunderbares Menschenwerk. Aus der Geschichte der mechanischen Automaten, Leipzig 1989

Scharf, Achim, Speicherprogrammierbare Steuerungen : Mehr Leistung und Komfort, in: Hard and Soft 6 (1989) 7/8,S. 8-15

Schmid, Wolfgang, Untersuchung der Arbeitsspiele der verschiedene selbsttätigen Steuerungen im Fertigungswesen, in: Feinmechanik und Präzision 49 (1941) 6, S. 65-69

Schmid, Wolfgang; Olk, Friedrich, Fühlergesteuerte Maschinen, Essen 1939

Stibitz, George R., The Organization of Large Scale Calculating Machinery, in: Harvard University Computation Laboratory (Hrsg.), Proceedings of a Symposium on Large Scale Calculating Machinery, Sponsered by the Navy Department Bureau of Ordnance and Harvard University at the Computation Laboratory Jan. 1947, Cambridge, Mass. 1948; wiedergedruckt in: in: Proceedings of a Symposium on Large Scale Calculating Machinery, in: Charles Babbage Institute (Hrsg.), Reprint Series for the History of Computing Bd. 7 , London, Los Angeles, San Francisco, S. 91-100

Strauch, Helmar, Selbsttätige Steuerung mechanischer Bewegungen durch Lochkarten, in: Maschinenbau 5 (1937) 9, S. 476-478

The ENIAC, Vol I. A Report Covering Work until December 1943, University of Pennsylvania, Moore School of Electrical Engineering, Philadelphia 1943

Torres y Quevedo, Leonardo, Essais sur l'automatique. Sa définition. Étendu théorique de ses applications, in: Revue de l'Académie des Sciences de Madrid, 1914; wiedergedruckt in: Revue Générale des Sciences Pures et Appliquées, 15.11.1915, S. 601-611; übersetzt in: Essays on Automatics. Its Definition - Theoretical Extent of Its Applications (1914), in: Randell, Brian (Hrsg.), The Origins of Digital Computers. Selected Papers, Berlin, Heidelberg, New York 1973, S. 87-105

Turing, Alan M., Lecture to the London Mathematical Society on 20 February 1947; wiedergedruckt in: Carpenter, B. E.; Doran, R. W. (Hrsg.), A. M. Turing's ACE Report of 1946 and other Papers (Charles Babbage Institute, Reprint Series for the History of Computing, Bd. 10), London, Los Angeles, San Francisco 1986, S. 106-124

Turing, Alan M., Proposal for Development in the Mathematics Division of an Automatic Computing Engine (ACE), presented to the National Physical Laboratory, 1945; wiedergedruckt in: Computer Science 57, National Physical Laboratory, Teddington 1972; wiedergedruckt in: Carpenter, B. E.; Doran, R. W. (Hrsg.), A. M. Turing's ACE Report of 1946 and other Papers (Charles Babbage Institute, Reprint Series for the History of Computing, Bd. 10), London, Los Angeles, San Francisco 1986, S. 20-105

Walther, Alwin, Moderne Rechenanlagen als Muster und als Kernstück einer vollautomatisierten Fabrik, in: Fritz Erler u. a. (Hrsg.), Revolution der Roboter, München 1956, S. 7-64

Wilkes, Maurice V., Early Programming Developments in Cambridge, in: Metropolis, Nicholas C.; Howlett, Jack; Rota, Gian-Carlo (Hrsg.), A History of Computing in the Twentieth Century. A Collection of Essays, New York, London 1980, S. 497-501