Zur Genese des informatischen Programmbegriffs:

Begriffsbildung, metaphorische Prozesse, Leitbilder und professionelle Kulturen

Hans Dieter Hellige, Universität Bremen

> "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 Analyse 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 Vorverständnishorizonten 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 mit spezifischen hermeneutischen Methoden und Konzepten rational rekonstruierbar. 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 sowie spezielle professionelle Kulturen. Deren Zusammenspiel soll im Folgenden am Beispiel metaphorischer Prozesse bei der Entstehung der Programmkonzepte untersucht werden.

Metaphern bilden 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, so meine Ausgangsthese, 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 kann der Lösungsraum u. U. von vornherein eingeengt werden.

2. Zur Bedeutung metaphorischer Übertragungsprozesse in der Wissenschaftsgenese der Informatik

Ein großer Teil informatischer Begriffe ist durch metaphorische Übertragungen aus anderen Technikbereichen oder Wissenschaften hervorgegangen. Das Metaphernsortiment der Informatik ist sogar, dies haben Richard Lynch (1993), Peter Mambrey, Michael Paetau, August Tepper und Carsten Busch (1998) betont, besonders bunt gemischt. Deren Erforschung hat sich bisher auch in informatischen und techniksoziologischen Analysen vor allem an geisteswissenschaftliche Methoden angelehnt. Metaphern wurden vor allem als Kommunikationsformen und -medien gesehen, bei der Interpretation aus dem Kontext gelöst und wie literarische Metaphern oder wie rhetorische Figuren in Kommunikationsprozessen gedeutet. Dies mag bei bildhaften oder unmittelbar einsichtigen Gestaltmetaphern wie Baum, Stapel, Schleife, Sprung und Bug, Virus oder bei (auto)suggestiven Bildsymbolen in Technikhypes 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 *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. Dies kann, wie bei Pflüger (u. a. 2002, 2003a/b)

anhand einer Kette epochaler Leitmetaphern mit epistemischen Charakter geschehen oder, wie im Folgenden, anhand einer vergleichenden Untersuchung von Metaphernbildungen zu einzelnen zentralen Begriffen und Konzepten der Informatik. So möchte dieser Beitrag am Beispiel der Entstehung des Programmbegriffs 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, Programmiersprache 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, mittelalterliche Uhrwerke, frühneuzeitliche Automaten und Spielwerke ebenso verwendet wie für moderne Computer. Durch die Allgegenwart des Programmbegriffs wird jedoch verdeckt, mit welcher Begrifflichkeit man jeweils arbeitete und warum man Jahrhunderte lang ohne einen von den jeweiligen materiellen Speichermedien Walzen, Lochkarten oder Stecktafeln losgelöste abstrahierende Bezeichnung ausgekommen ist. Denn für Babbage, Ludgate und Torres 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.

So bezeichnet Babbage, dem „mettre en carte" bei Jacquard entsprechend, Programmiervorgänge meistens als „arrangement" oder

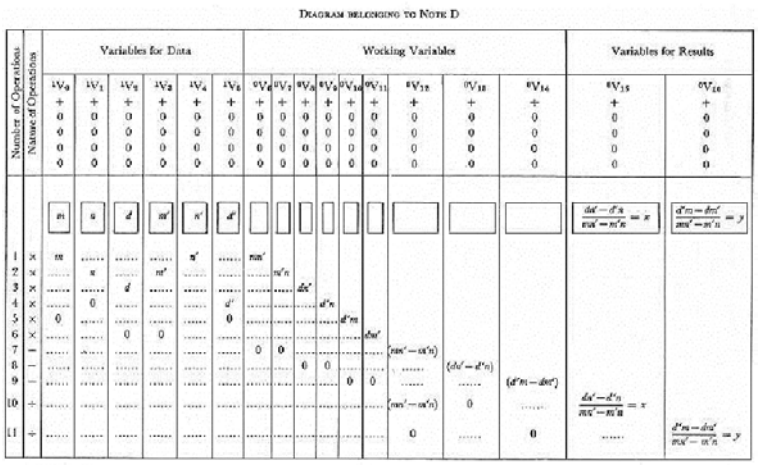


DIAGRAM BELONGING TO NOTE D

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
		n_1	n_2	n_3	n_4	n_5	n_6										$\frac{da'-d'a}{ea'-a'e}=x$	$\frac{d'e-de'}{ea'-a'e}=y$
1	×	n_1	.....	.....	.....	n_5	.....	n_1n_5										
2	×	.....	n_2	.....	n_4	.....	.....	.....	n_2n_5									
3	×	.....	.....	n_3	.....	.....	.....	.....	.....	dn_5								
4	×	.....	0	.....	.....	d_5	.....	.....	.....	.....	d_5n							
5	×	0	.....	0	.....	0	.....	.....	.....	.....	.....	d_5n						
6	×	.....	.....	0	0	.....	.....	.....	.....	.....	.....	.....	dn_5					
7	−	.....	.....	.....	.....	.....	.....	0	0	.....	.....	.....	.....	$(n_1n_5-n_2n_4)$				
8	−	.....	.....	.....	.....	.....	.....	.....	.....	0	0	.....	.....	.....	(dn_5-d_5n)			
9	−	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	0	0	.....	.....	$(d_5n_1-dn_2)$		
10	÷	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	$(n_1n_5-n_2n_4)$	0	.....	$\frac{da'-d'a}{ea'-a'e}=x$	
11	÷	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	0	.....	0	.....	$\frac{d'e-da'}{ea'-a'e}=y$

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

„combination" der verschiedenen „sets of cards" zu einer „order by means of cards" bzw. „chain of operational cards" (Babbage 1937, S. 17 ff. bes. S. 45 f.; Merrifield 1879, S. 57). In einer Notiz vom Juli 1836 spricht er davon, dass „cards (Jacquards) of the Calc. engine direct a series of operations", wobei in den Lochmustern „small pieces of formulae" enthalten sind (zit. nach Randell, S. 349). 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 1842). Schließlich bedienen sich Ada Lovelace und Babbage der viel zitierten Analogien zu den Jacquard-Webstühlen, 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). Ähnlich schreibt Babbage in einem Brief an Arago im Dezember 1839 „we can communicate to a very ordinary loom orders [sic!] to weave any pattern that may be designed. [...] Availing myself of the same beautiful invention, I have by similar means communicated to my calculating engine orders to calculate any formula however complicated [...]". Doch im Gegensatz zu Ada streicht er den Unterschied der 'Programmierung' von Jacquard-Maschinen und der Analytical Engine heraus: „[...] but I have also advanced one stage further, and I have communicated through the same means orders to *follow certain laws in the use of those cards* [...]" (zit. in Merrifield 1879, S. 57, meine Hervorhebung).

Der Übergang von der Mechanik zur Elektromechanik nach 1900 ändert an den Programmbegriffen und -Metaphern zunächst nur wenig. Percy E. Ludgate wandelt mit seinen Lochstreifen nur den Ablauf der Bedien- und Rechenprozesse ab, bleibt aber ansonsten noch ganz der Babbage-Terminologie verpflichtet und greift einmal sogar auf die Ada-Metaphorik zurück (Ludgate 1909). Leonardo Torres y Quevedo lehnt sich bei dem Programmier- und Steuerungskonzept seiner elektromechanischen „Analytischen Maschine" z. T. an Babbages Begriff des „Arrangements" von Operationen an und beschreibt dieses in seinen einzelnen Berechnungsschritten. Dabei sieht er bereits die Zusammensetzung komplexerer Arbeitsfolgen aus Elementaroperationen vor (Torres 1914, S. 100). Doch er entwickelt andererseits ein anthropomorphes Verständnis des ‚Programmablaufs', wonach das hypothetische „automaton" sich wie ein intelligentes Wesen den jeweiligen Bedingungen anpasst und bei der

Pfadwahl selber Entscheidungen trifft: „In essence, the automaton acts as a circumspect person and reflects: it examines the present circumstances in order to decide what it should do and then it does it" (Torres 1920, S. 116). Beim Fortschreiten der einzelnen Operationen der Formelberechnung folgt es zuvor aufgestellten Regeln bzw. *Verhaltensmustern*, die das Arbeitsverhalten präzise festlegen. Dabei geht Torres sogar schon von einer Erstellung des ‚Programms' in natürlicher Sprache aus.

Die Steuerung („contrôle") der Maschine wird auf diese Weise sehr einfach: Der „human calculator who uses the machine" muss nur von Zeit zu Zeit bestimmte Tasten drücken und selbst dies könne noch automatisiert werden durch „a process similar to that which has been used for operating a *mechanical piano*" (Torres 1914, S. 88 f., Zitate in engl. Übersetzung S. 95, meine Hervorhebung). Torres stellt mit dieser Metapher eine direkte Verbindung zwischen seiner theoretischen Automatologie und der Welt der Lochstreifen-gesteuerten Musikautomaten her. Er erkennt, dass die Verallgemeinerung der ‚Programmsteuerung' alle Maschinen zu potentiellen Automaten macht. „These very principles may be applied to other calculating machines and even to industrial machines" (Torres 1920, S. 116). Doch den Schritt zur Abstraktion eines Hardware-unabhängigen Programmbegriffs geht Torres noch nicht, dazu versperrte ihm sowohl die Fixierung auf die traditionelle Automatenwelt, ihre Mechanismen und Steuerungen als auch die anthropomorphe Sichtweise intelligenter Automaten den Weg.

Wie und wann der Loslösungsprozess beginnt und welche Rolle die Programmmetapher dabei spielt, ist noch ziemlich im Dunkeln. Die Forschung stützt sich in der Regel ausschließlich auf das Oxford English Dictionary, das die frühesten Belege bei Computern für 1945 ausweist (siehe u.a. Beniger 1986, S. 39, Coy 1998). Nach Paul E. Ceruzzi (1998, S. 20 f.) liegt der Ursprung des Programmbegriffs des Computerbereichs beim ENIAC-Team der Moore-School, David Alan Grier verweist ausdrücklich auf John Mauchlys berühmtes Memorandum von 1942 „Use of High Speed Vacuum Tube Devices for Calculating", das die ENIAC-Genese einleitete. In Studien über Konrad Zuse werden sogar Manuskripte zitiert, in denen der Programmbegriff angeblich schon seit 1937 vorkommt. Doch diese Datierungen sind wenig glaubwürdig, da in diesen Dokumenten bereits von der „Minimalform der rein universellen Programmiersprache", „allgemeinen Formalsprachen", „algorithmischen Sprachen" und vom „bedingten Sprung" die Rede ist, und zwar nicht nur

in der später angefertigten Transskription, sondern bereits in der stenografischen Notiz!

Zuse hat zwar schon seit dem Ende der 30er Jahre ein sehr avanciertes ‚Programm'konzept und eine informatische Sichtweise entwickelt, die wissenschaftliche Probleme durch mathematische Modellbildung und Programmierung durchgängig rechnerunterstützt lösen will (vgl. hierzu bes. H. Zuse 2003; zu den Datierungsproblemen des Zuse-Nachlasses Hellige 2003, S. 415 f.). Doch hielt er bis in die zweite Hälfte der 40er Jahre an dem in Rechenbüros bereits Anfang der 30er Jahre üblichen Begriff „Rechenplan" fest. Er definierte den „Rechenplan" als „Aufführung der aufeinanderfolgenden Rechenoperationen", doch vielfach verwendete er ihn auch einfach synonym mit „Lochstreifen": „Rechenpläne sind Lochstreifen, die von den Geräten abgetastet werden." (Zuse 1936, Zuse 1946, S. 2) Sehr weitreichend ist seine Unterscheidung von Gesamtplan, der einer Formelsammlung bzw. einem Planbestand entspricht, von aufgabenspezifischen Plangruppen und ganz speziellen Einzelplänen, die z. T. aus Teilen des „dauernden Planbestands" zusammengesetzt sein können. Er entwickelte 1942-46 die Vorform einer Programmiersprache, den „Plankalkül" (Zuse 1945), doch bezeichnete er diesen erst nach Abschluss der Arbeiten als „eine allgemeine mathematische ›Zeichensprache‹", mit der man „Anweisungen" für die unterschiedlichsten „schematischen kombinatorischen Denkoperationen" mechanisieren könne (Zuse 1947, S. 1 f.). Obwohl sein ‚Programm'-Konzept, ohne dass der Begriff selbst auftaucht, in den späten 30er und frühen 40er Jahren den anderen Entwicklungen in Europa und den USA weit überlegen war, hatte es aufgrund der Isolation Zuses keinen Einfluss auf die Entstehung des Programmbegriffs. Deshalb wird es im Folgenden auch ausgeklammert.

3. Programmbegriffe in den professionellen Kulturen der Steuerungs- und Regelungstechnik und der Lochkartentechnik

3.1 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 Betätigungsvorgänge der Maschine" (Meiners 1936, 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 (SPS) Richard Morley als Vorbild für die SPS-Programmiermethode des „Kontaktplans" („ladder diagram") gewählt wurde (Meiners 1936, S. 35 ff.; 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 Zukunftsaussichten 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 zuvor 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" (De Leeuw 1922a, S. 641, meine Hervorhebung). 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 Einzweckautomaten waren, es entstand bereits die Idee einer *codebasierten* Mehrzwecksteuerung für unterschiedlichste Maschinen und Prozessfolgen. Dabei war ihm bewusst, dass es sich um eine ganz neuartige Maschine handelte: „I would call this style a ›*program machine*‹“ (ebenda). 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“ (ebenda, S. 642). Ob De Leeuw damit direkt an die automatologischen Visionen von Torres anknüpft oder ob er selber eine metaphorische Verbindung zu traditionellen Automatensteuerungen herstellt, bedarf noch weiterer Recherchen. Auf jeden Fall sind die Parallelen sehr auffällig, und es könnte durchaus sein, dass ihn gerade Torres' Hinweis auf die „automatisation“ industrieller Maschinen im „Bulletin de la Societé d'Encouragement pour l'Industrie Nationale“ von 1920 dazu veranlasst hat, dessen Überlegungen weiterzuführen. Etwa, wenn er für die komplizierten Verknüpfungen einer ganzen Reihe von Werkzeugen mit einfachen, parallelen und wiederholten Bearbeitungszyklen eine Art Assemblersprache entwickelte, 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 wohl 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 ff.).

Mit dem Programm- und Befehlsbegriff und der Codesteuerung ging De Leeuws „program machine“ von 1920-22 über die bekannten programmgesteuerten Maschinenkomplexe der 20er bis 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

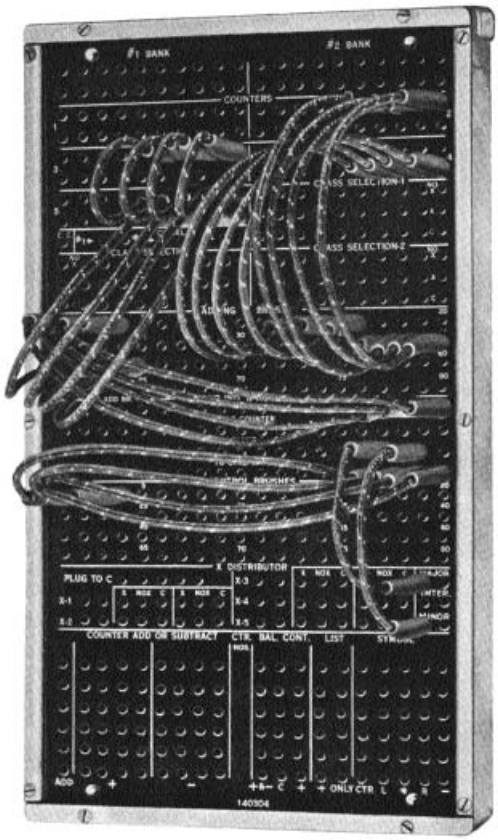
und Lochbandsteuerung bei den automatischen Klavieren. Es hat den Anschein, dass die Metapher der Lochband-gesteuerten automatischen Klaviere nicht nur wie seinerzeit die Jacquard-Steuerung für Babbage die Konstruktionsidee geleitet, sondern auch den Anstoß für die Verwendung des Programmbegriffs geliefert hat. Jedenfalls leitet sich der informationstechnische Programmbegriff damit ursprünglich nicht aus dem Radioprogramm ab, wie man meist annimmt, sondern aus der Welt der Musikautomaten. 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 bis 30er Jahren die steuerungs- und regelungstechnische Konstruktionstradition des „program control". Ihr Leitbild war nicht mehr die virtuose Handhabung der physikalischen Verknüpfungen wie bei den Vertretern des „manuellen Programmierens" (vgl. Hellige 1998), sondern eine auf Automatisierung zielende Zusammenführung der Programmierschritte in „program devices".

3.2 Das Programmkonzept bei verketteten Lochkartenmaschinen: Das "setup" mit "automatic plugboards" bei Wallace Eckert

In den späten 20er und vor allem in den 30er Jahren erscheint das Konzept einer zentralen Programmier- und Kontrollschnittstelle auch im Bereich der Lochkartenmaschinentechnik sowie bei Rechen- und Buchungsmaschinen. Die Schalt- bzw. Stecktafel wurde zwar schon 1895 von Otto Schäffler als „logischer Vermittlungsschrank" auf der Basis der Metapher des Telefon-Plugboards erfunden und war ab 1905 auch Bestandteil von Hollerith-Maschinen (Zemanek 1988, S. 21). Doch erst mit der Verkettung unterschiedlicher Lochkartengeräte zu Maschinenkomplexen entwickelte sie ihre ganze Funktionsbreite. Aus den ursprünglich nur für Zähl- und Sortierzwecke geschaffenen Lochkartenmaschinen entstanden so multifunktionale Rechen- und Informationsverarbeitungsmaschinen. Durch die Einführung des „removable" bzw. „automatic plugboard" in der IBM 601 von 1934 wurde sogar die Speicherung, Auswechslung und Weitergabe der Steck'programme' möglich (vgl. Hellige 1998, S. 191 f.).

Bahnbrechend war hierbei besonders Wallace J. Eckert, der ab 1933/34 für astronomische Berechnungen verschiedene Lochkartenmaschinen zu einem Maschinenkomplex verkoppelte. Sein Ziel war ein mög-

lichst vollautomatischer Ablauf komplizierter wissenschaftlicher Berechnungen, um den Menschen als Fehlerquelle auszuschalten: „For this work it must be possible to change quickly from one complicated *setup* to another. The *calculation control switch* controls the operation of the multiplier, tabulator, and summary punch so that a cycle of different arithmetical operations can be performed in rapid succession" (W. Eckert 1940, S. 22, meine Hervorhebung). Das Plugboard des „*calculation control switch*" war über „rotating cams" und „multicontact relays" mit den jeweiligen Speziallochkartenmaschinen verbunden, wodurch die Verdrahtung der Plugboards effizient variiert werden konnte (ebenda, S. 76 f.). Die Gestalt und die Bedienung der jeweils aus mehreren Lochscheiben gebildeten Steuerwalzen folgte dabei der Metapher traditioneller Musikautomaten: „A series of about twenty of these disks are attached to a common shaft to form a sort of player piano roll" (ebenda, S. 77).



Programmierschnittstelle für Wallace Eckerts verkettete Lochkartenmaschinen (1940, S. 14)

Wallace Eckert ging mit dieser äußersten Ausreizung der Möglichkeiten der Lochkartentechnik einen großen Schritt in Richtung eines frei programmierbaren Rechners (Kistermann 2000). Doch im Unterschied zu De Leeuw gelangte er trotz ähnlicher Metaphern und ‚Hardware-/Software'-Arrangements nicht zu einem von den physikalischen Medien unabhängigen Programmbegriff. Er blieb in seiner Begrifflichkeit vielmehr noch weitgehend der alten Lochkartenwelt verhaftet. Über die darin üblichen Bezeichnungen „plugging", „wiring", „setup" und „arrangement" hinaus findet sich bei ihm vor 1948 kein abstrahierender Programmbegriff. Nur an einer Stelle seines Hauptwerkes von 1940, das seine Arbeiten der 30er Jahre zusammenfasst, heißt es: „The planning of an extensive *program* thus requires a careful analysis of many factors in the light of all available data and experience" (W. Eckert 1940, S. 25, meine Hervorhebung). Doch scheint vom Kontext her hier mit „program" eher ein Forschungsprogramm gemeint zu sein.[1] Auch die in der Literatur (Beniger 1986, S. 401; Weinhart 1990, S. 148) genannte Bezeichnung „*mechanical programmer*" für den „calculation control switch" ist wohl eher eine nachträgliche Begriffsschöpfung, jedenfalls taucht sie in den Eckert-Schriften vor 1945 nicht auf. Gleichwohl gingen von diesem Konzept der Plugboard-Programmierung wichtige Impulse auf die ENIAC-Entwicklung aus.

1) "Program" ist auch im ausführlichen Register aller technischen Fachbegriffe des Eckert-Bandes nicht enthalten und kommt im Text auch sonst nur in Verbindung mit Forschungsprogrammen vor (ebenda, S. 79 ff.).

3.3 Weiterwirken des "program control" Leitbildes bei John P. Eckert und John Mauchly: Automatisierung durch Integration der Programmierschritte in "program devices"

Auch beim ENIAC 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 die professionelle Kultur des „program control" erklärt sich auch das auffällig kontroll- und automatisierungstechnische Verständnis des Programmierens bei den beiden aus der Elektrotechnik kommenden Chefdesignern John Mauchly und John Presper Eckert. In den ENIAC-Proposals und -Berichten von 1942-44 erscheint der Programmbegriff nämlich fast ausschließlich in Verbindung mit „devices", „circuits", „pulses" oder „switches", 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 calcula-

tion in their proper order. The program control unit can be equipped, if desired, with a punch-card program selector to facilitate rapid set-up of different problems" (Mauchly, Eckert, Brainerd 1943, zit. nach Burks 1980, S. 335 f.). Auch in den Berichten, Memoranden und Vorträgen der Jahre 1946/47, 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 auf Vorschlag der Von-Neumann-Gruppe 1944 zu der bis dahin dezentralen Kontrollstruktur hinzukam, verbirgt sich nicht etwa der Chefprogrammierer, sondern ein „central control switchboard", an dem durch Steckverbindungen die lokalen „controls" der Unterprogramme zu einem „single program" sequenziert wurden: „We propose a centralized programming device in which the program routine is stored in coded form. 'The' crucial advantage of central programming is that any routine, however complex, can be carried out whereas in the present ENIAC we are limited" (Brief von H. H. Goldstine an J. von Neumann vom 2.9.1944, zit. nach Macrae 1992, S. 284; A. Goldstine 1946).

Der ENIAC-"Master-Programmer", Panel 2 (www.kondo3d.com/eniac / DSCF2453.jpg)

Mit dem Begriff „program routine" deutet sich zwar die Loslösung des abstrakt-logischen Programms von der Hardware an, doch in der überwiegenden Zahl der Belege, vor allem bei Eckert und Mauchly selber, fielen „program" und „programming" noch immer mit der Gesamtheit der Kontrollgeräte und technischen Steuerungsprozesse zusammen.

Die Bindung an das steuerungs- und regelungstechnische bzw. automatologische 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": 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 scales 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, meine Hervorhebung).

Bei den Moore-Lectures propagierte auch 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" im umfassenderen Sinne einer automatischen Programm-Generierung[2] 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", der „Automatologie" und der in ihrer Tradition stehenden ENIAC-Entwickler herausgearbeitet wurden, sondern von Mathematikern wie John von Neumann und Alan Turing, die sowohl praktisch wie theoretisch in die Rechner- und Programmkonstruktion involviert waren.

2) Die von Elbourn und Ware lebhaft beklagte ständige Verwechslung bzw. Gleichsetzung von "automatic coding" und "automatic programming" (1962, S. 1059 f.) war ein Kennzeichen dieser Fehleinschätzung. Zu den unterschiedlichen Konzepten des "automatic programming" siehe Sammet 1969, S. 4, 13.

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

4.1 ‚Programmieren' als konstruktives Planen und Problemlösen in der "Von-Neumann-Gruppe"

John v. Neumann entwickelte mit Hermann H. Goldstine und Arthur Burks zusammen 1945-48 ein Sechsstufenmodell der Programmiervorgänge, wobei klar zwischen den nicht automatisierbaren konzeptionellen, konstruktiven und dynamisch-analytischen Aufgaben einerseits und eher routinisierbaren statischen Codierungsprozessen andererseits unterschieden wurde (Aspray 1990, S.70). Mit Blick auf die Mannigfaltigkeit der Programmiertätigkeiten verzichtete von Neumann, der ansonsten mit einer ganzen Reihe von Analogiebildungen aus Natur-und Technikwissenschaften experimentierte, hier auf eine technische, biologische oder anthropomorphe Leitmetapher. Zwar wird der Code-Begriff, der teils für das Gesamtresultat der Programmerstellung, teils auch für deren Umsetzung in Maschinencode steht, in der „Preliminary Discussion" gelegentlich mit der Sprach- und Übersetzungsmetapher belegt: „[...] problems can be coded, i. e. prepared in the language the machine can understand [...] a unit which can understand these instructions and order their execution."[3] Doch 1947 wandten er und Goldstine sich deutlich gegen die Vorstellung einer bloßen Übersetzung. Die Annahme, Coding bedeute „Translating a meaningful text [...] from one language (the language of mathematics, in which the planner will have conceived the problem) into another language (that one of our code)" sei falsch: „Thus the relation of the coded instruction sequence to the mathematically conceived procedure of (numerical) solution is not a statical one, that of a translation, but highly dynamical [...] Since coding is not a static process of translation, but rather the technique of providing a dynamic bakkground 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" (Goldstine, von Neumann 1947, S. 82 f.). So wird in diesen Schlüsseltexten zwar die Sprachmetapher für die Programmierung eingeführt, doch bei der ausführlichen Behandlung der einzelnen Prozesse wieder explizit zurückgenommen.[4]

Es findet sich nicht einmal ein einheitlicher Begriff für die Gesamtheit der Programmiervorgänge, ja von Neumann weigerte sich

3) Burks, Goldstine, von Neumann 1946, S. 34 f.; siehe auch Goldstine, von Neumann 1946, S 30 :"translate the problem (once it is logically reformulated and made explicite in all its details) into the code."

4) Forschungsansätzen, die auf Anthropomorphismen und Mensch-Maschine-Hybride in der Von-Neumann-Gruppe fixiert sind, entgeht dieser insgesamt sehr vorsichtige Umgang mit derartigen Metaphern (vgl. Eulenhöfer 1998 und Stach 1998).

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.[5] Und wenn, dann auch nur in der partiellen Bedeutung der Bildung von Funktionen aus im Rechner gespeicherten Subroutinen für Addier-, Gleitkomma- und dergleichen Operationen.[6] Mit den Stufen Planung und logische Problemlösung als übergeordneten Tätigkeiten und der Formulierung des Codes als ausführende Detailarbeit 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 mit dem damals noch stark kontrolltechnischen Verständnis des Programmbegriffs nicht vereinbaren.

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

6) So heißt es in der "Preliminary discussion": "these operations can be programmed by means of others."; "programming it out of operations built into the computer"; "programmed as subroutines out of orders already incorporated in the machine" (Burks, Goldstine, von Neumann 1946, S. 70, 74, 77).

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 ff.). 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 wohl 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-Sicht" 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, von 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" (ebenda, S. 1).

Obwohl durch und durch Mathematiker, reflektierte von Neumann Computer- und Programmstrukturen als komplexes, zielkonfliktbehaftetes Konstruktionsproblem, für das ihm die Planungs- und

Organisationsmetapher adäquater erschien. Eine sehr ähnliche Design-Auffassung veranlasste um 1960 Frederick Brooks, den Computer Engineering-Begriff durch die Architekturmetapher zu ersetzen (Hellige 2003, S. 436 ff.). Mit *Organisation*, *Planung* und *Architektur* hat die Informatik reflexiv-soziomorphe Metaphern ins Spiel gebracht, die sowohl auf die Arbeitsprozesse der Konstruktion, die Stakeholder-bezogene Zielkonfliktstruktur und die daraus resultierende Design-Komplexität dieser technischen Gebilde verweisen. Der Wechsel zu sozialen Metaphern, die nicht wie später die „Hierarchien", „Pyramiden" und „Fabriken" vorab bestimmte soziale Strukturen festschreiben wollen, signalisiert hier, zumindest in der Entstehungsphase der Metaphern, ein Sicheinlassen auf die technisch-soziale Komplexität konstruktiver Prozesse, die bei den anfänglich dominierenden technischen und naturwissenschaftlichen Metaphern noch ausgeblendet war.

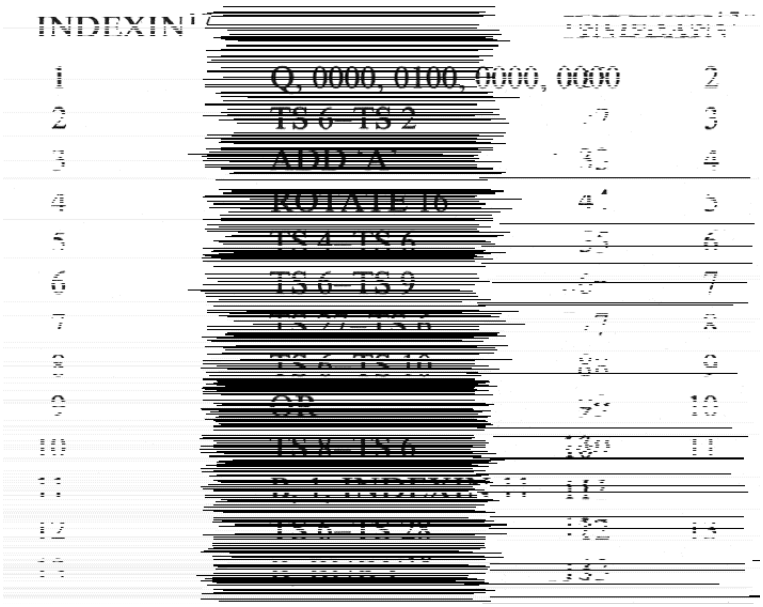
4.2 Programmieren als Überbrücken von Sprachdifferenzen zwischen Mensch und Maschine bei Turing

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 als ein Problem der Sprachdifferenz gedeutet. Die bei von Neumann nur angedeutete und später zurückgenommene Sprachmetapher wird bei Turing Grundlage des Verständnisses des Programmierens. Die Lücke zwischen der symbolischen Sprache der Maschinen und der Alltagssprache des Menschen ist für ihn wesentlich ein Kommunikationsproblem: „It should 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). Turing entwarf daher bereits in seinem "Proposal" unter-

schiedliche Sprachebenen für die „instruction tables", mit denen die „jobs" [sic!] abgearbeitet werden: die „machine form", d.h. der Maschinen-Code, die „permanent form", die Lochkarten-Notation und die „popular form", eine symbolische Sprache, „which can easily be read". Die in verschiedener Form gesammelten Programme sollten nach klassischer Manier in

7) Die "Library"-Metapher gibt es aber schon in der Zeit der Jacquard-Maschinen und bei Babbage.

"Popular form" einer Instruktionstabelle in Turings "Proposal" von 1945 (1946, S. 75)



„a sort of library" aufbewahrt und geordnet werden[7] (Turing 1946, S. 70 ff., bes. 74).

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'. So kehrte die

Metapher zum programmierenden Subjekt zurück und löste sich damit auf. 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, die „girls" erfassten die Werte und Daten und die „servants" fütterten den Rechner mit Lochkarten. 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 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" (ebenda, 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.). Auch 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 Arbeitsprozesse der Programmierung in den Blick bekommen. Er befreite damit die Begriffe „program" und „programmer" von der Hardware-Fixierung in

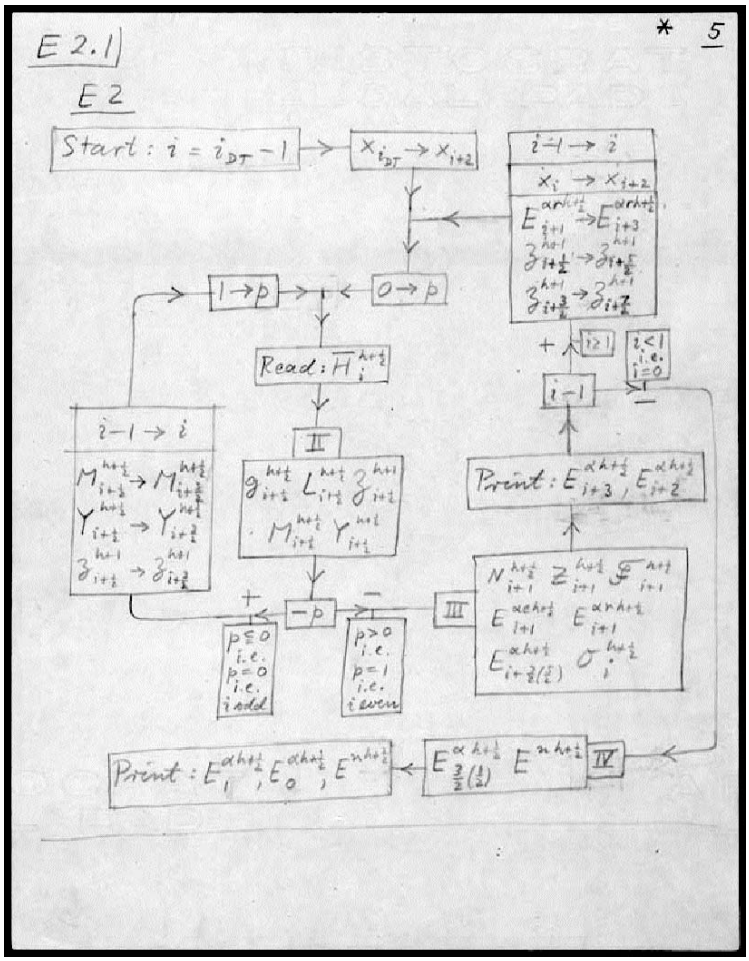
der professionellen Kultur des „program control". Er entwickelte schon 1945, d. h. noch vor der von-Neumann-Gruppe, wohl als erster ein Sprachkonzept der Programmierung, allerdings noch ohne den Begriff der Programmiersprache. Dessen erstes Erscheinen ist wie das der Begriffe Betriebssystem und Software 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.

5 Die Anfänge von Rationalisierungsmetaphern in der Programmierung und das Problem von metaphorischen Prozessen mit unreflektierter Leitbildfunktion

John von Neumann und Goldstine haben beiden Tendenzen widerstanden, dem Glauben an eine schnelle Realisierung des „automatic programming", wie an eine baldige Überwindung der Sprachdifferenzen zwischen Mensch und Maschine. Sie bewahrten sich aus ihrer 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 sie 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" (Goldstine, von Neumann 1947, S. 84).

Das adäquate Modellierungsinstrument für diesen dynamischen Ablauf der Berechnung und die Verschachtelung der Programmteile war für die von-Neumann-Gruppe seit Sommer 1946 das graphische „Flussdiagramm", das als „Ablaufschema" oder „Strukturdiagramm" traditionell in den Ingenieurwissenschaften zur Darstellung von komplexen Anlagenstrukturen sowie von Stoff- und Energieflüssen verwendet wurde (Goldstine 1972, S. 266 ff.; Knuth, Pardo 1980, S. 208 ff.). Diese „*flow diagrams*" wurden erst im Laufe der 50er und frühen 60er Jahre von den „*flow charts*" abgelöst, die sich im Namen und in der Symbolik an die Flowcharts der beiden Gilbreth anlehnten. Die Flowcharts dienten im Computing ursprünglich nur zur Veranschaulichung von Arbeits-

und Informationsflüssen, doch wurde diese Darstellungsform auch bald auf Programmabläufe übertragen, so dass sich am Ende die Unterschiede der Notationen verwischten (siehe die Klage darüber bei Chapin 1962, S. 97). Dieser Wandel der Darstellungsmodelle steht aber bereits im Zusammenhang mit dem breiten Einzug industrieller Metaphern und Modellbildungen in diesem Zeitraum. Es ging jetzt nicht mehr vorrangig



Flowdiagram von John von Neumann

um die Darstellung und Planung komplexer Strukturen, sondern um eine *Rationalisierung* der Softwareproduktion.

Process Charts der Gilbreth als Vorbild der Flowcharts (Barnes, S. 57)

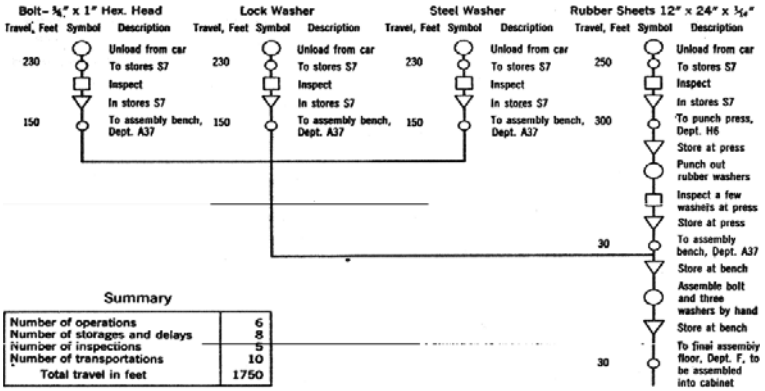


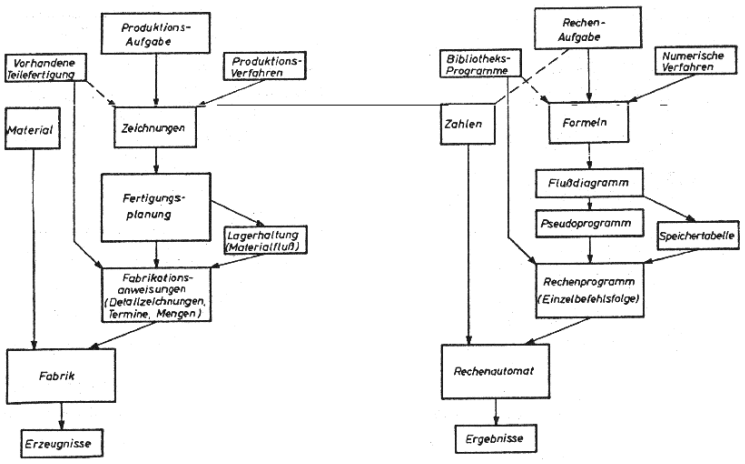
Fig. 32. Assembly process chart—bolt and washer assembly.

Dies wird nicht zuletzt daran deutlich, dass man auch für die Strukturierung des Software-Entwicklungsprozesses ab 1962/63 ebenfalls auf tayloristische Rationalisierungsinstrumente 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 gegangen war, dass es sich um *konstruierte* Analogiebildungen mit begrenzter Reichweite handelt, konnte es leicht zu metaphorischen Zirkelschlüssen kommen. So bereits bei einer der frühesten Analogiebildungen von Programmierung und industrieller Fertigungsplanung, der 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; vgl. auch Eulenhöfer 1998, S. 260 f.). Letzte-

re verglich er auch mit dem Fließband, während das „Kommandowerk" das „Analogon zur Betriebsleitung" darstelle (ebenda, S. 38). Auch das noch so mühsame und langwierige Programmieren unterwarf er der ‚Fabrikordnung': Die einzelnen Prozesse und Phasen wurden exakt der Fabrikationsplanung nachgebildet.

Dabei ließen die gleichzeitige Modellierung der Programmfertigung als Abbild der industriellen Fertigungsplanung und die Deklarierung des Rechners und seiner Programmstruktur als Vorbild für die Fabrikorganisation Ursprungs- und Zielbereich der Metaphernbildung verschwimmen, und es kam zum metaphorischen Zirkelschluss. Wenn aber die soziomorphe Modellierung von Technostrukturen und die technomorphe Modellierung von Sozialstrukturen sich gegenseitig verfestigen, dann erhalten sie undurchschaubaren Ideologiecharakter, ein Phänomen, das sich später bei den hierarchischen Pyramiden- und Referenzmodellen bestätigen sollte (vgl. die Kritik bei Hammacher 1996, S. 88 ff.). Man sollte deshalb immer die Ambivalenz von Metaphern im Blick behalten: die Vorprägung durch Bestehendes und die kreative Neuschöpfung.



Alwin Walthers industrielle Programm-Metapher von 1956 (S. 41)

Bild 15

Vergleich zwischen der Planung einer Fabrikation und dem Programmieren eines Rechenablaufs

6 Fazit: Aufgaben für die Metaphernforschung in der Technikbewertung

Aus dem hermeneutischen Charakter von Übertragungsprozessen ergibt sich, so meine These, eine veränderte Sicht der Rolle der Metaphernforschung in der Technikbewertung. Metaphern 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 wenig über die systemische Problemstruktur und die Langzeitdynamik einer Technik aus.

Eine mit historischen Vergleichen arbeitende hermeneutische Bewertung aktueller Techniken vermag zwar selbst 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 wichtige Aufgaben zu leisten. Denn die Vorurteilstruktur des Verstehens ist nicht, wie Gadamers Hermeneutik-Auffassung es nahelegt, 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“ (Habermas 1967, S. 175).

Literatur

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

Babbage, Charles (1837), *On the Mathematical Powers of the Calculating Engine*. In: Randell, Brian (ed.) (1973), 17–52

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

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

Burks, Arthur W.; Goldstine, Herman H.; Neumann, John von (1946), *Preliminary 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; nachgedruckt in: Taub, Abraham H. (ed.), Complete Works of John von Neumann, 6 Bde.Oxford (UK), New York 1961-63, Bd. 5, 34–79

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

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

Chapin, Ned (1962), *Einführung in die elektronische Datenverarbeitung*. Wien, München

Clippinger, Richard F. (1948), *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; Internet: http://ftp.arl.army.mil/~mike/comphist/48eniac-coding/

Coy, Wolfgang (1998), *Hat das Internet ein Programm?* Vortrag auf dem medienwissenschaftlichen Symposium der Universität Konstanz am 30.10.98, Internet: http://waste.informatik.hu-berlin.de/Coy/Coy_Internet_11-98.html

De Leeuw, Adolph L. (1922), *Methods of Machine Tool Design.* American Machinist 57 (1922) Heft 17, Oktober 1922, S. 639-642 (Schlussabschnitt: "The Program Machine and Its Future possibilities")

Eckert, John P. (1944), *Disclosure of Magnetic Calculating Machine*.29. Januar 1944, Kopie 1. Feb. 1945; gedruckt in: Lukoff, Herman From Dits to Bits: A Personal History of the Electronic Computer, Portland, OR. 1979, 207–209

Eckert, John P. (1946), *A Preview of a Digital Computing Machine, Lecture 10.* In: Campbell-Kelly, Martin; Williams, Michael R. (eds.), 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, 109–126

Eckert, John Presper; Mauchly, John W. (1945), *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 (1945), *Description on the ENIAC*. Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia

Eckert, Wallace J. (1940), *Punched Card Methods in Scientific Computation*. Thomas J. Watson Astronomical Computing Bureau, New York; wiedergedruckt in: Charles Babbage Institute, Reprint Series for the History of Computing, Bd. 5, London, Los Angeles, San Francisco 1984

Elbourn, R. D.; Ware, W. H. (1962), *The Evolution of Concepts and Languages of Computing*. Proceedings of the IRE 50, 1059–1062

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

Eulenhöfer, Peter (1998), *Der Informatiker als "deus ex mathematica"*. In: Siefkes, Dirk; Eulenhöfer, Peter; Stach, Heike; Städtler, Klaus (Hrsg.), Sozialgeschichte der Informatik. Kulturelle Praktiken und Orientierungen, Wiesbaden, 257–273

Gilbreth, Frank B.; Gilbreth, Lilian M. (1921), *Process Charts*. Transactions of the ASME 43, Paper 1818, 1029–1050

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

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

Goldstine, Herman H.; Goldstine, Adele (1946), *The Electronic Numerical Integrator and Computer (ENIAC).* Mathematical Tables and other Aids to Computation 2 (1946) Juli, 97-110; nachgedruckt in: Randell, Brian (ed.) (1973), 333-347

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

Goldstine, Herman H.; Neumann, John von (1947/48), *Planning and Coding of Problems for an Electronic Computing Instrument, 3 Bde.* Institute for Advanced Study, Princeton 1947–1948; wiedergedruckt in: Taub, Abraham. H. (ed.), Complete Works of John von Neumann, 6 Bde. Oxford (UK), New York 1961-63, Bd. 5, Part II, Volume 1: 80–151; Volume 2: 152–214; Volume 3: 215–235; und in: Aspray, William; Burks, Arthur W. (eds.), Papers of John von Neumann on Computing and Computing Theory, Cam-

bridge / Mass., London, Los Angeles, San Francisco 1987, Volume 1: 151–222; Volume 2: 223–285; Volume 3: 286–306

Grier, David A. (1996), *The ENIAC, the Verb „to program" and the Emergence of Digital Computers*. Annals of the History of Computing 18 (1), 51-55

Habermas, Jürgen (1967), *Zur Logik der Sozialwissenschaften*, in: Philosophische Rundschau, Beiheft 5, Tübingen

Hammacher, Bernd (1996), *Referenzmodelle als Leitbilder der CIM-Gestaltung.* In: Hellige, Hans Dieter (Hrsg.), Technikleitbilder auf dem Prüfstand. Das Leitbild-Assessment aus Sicht der Informatik- und Computergeschichte, Berlin, 71–95

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

Hellige, Hans Dieter (1996a), *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, 13–36

Hellige, Hans Dieter (1996b), *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, 205–234

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

Hellige, Hans Dieter (2003), *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, 411–470

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

Kistermann, Friedrich W. (2000), *The DEHOMAG D11 Tabulator - A Milestone in the History of Data Processing.* In: Rojas, Raúl; Hashagen, Ulf (eds.), The First Computers - History and Architectures, Cambridge, MA, London, 221–235

Knuth, Donald E.; Pardo, Luis Trabb (1980), *The Early Development of Programming Languages, The Early Development of Programming Languages*. In: Metropolis, Nicholas C.; Howlett, Jack; Rota, Gian-Carlo (eds.), A History of Computing in the Twentieth Century. A Collection of Essays. New York, London, 197–273

Kröck, Alwin (1991), *SPS ‚der Schlüssel zur Automatisierung.* Technische Rundschau, 47, 68–73

Ludgate, Percy E. (1909), *On a Proposed Analytical Machine.* Scientific Proceedings of the Royal Dublin Society 12 No. 9, 77–91; nachgedruckt in: Randell, Brian (ed.) (1973), 71–85

Lynch, Richard K. (1993), *On Analytical 'Engines', Data 'Architectures' and Software 'Engineers': Metaphoric Aspects of the Development of Computer Terminology*". Ph.D. Thesis, Columbia University Teachers College

Macrae, Norman(1992), *John von Neumann*. New York

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

Mauchly, John W. (1942), *The Use of High Speed Vacuum Tube Devices for Calculating.* nachgedruckt in: Randell, Brian (ed.) (1973), 329–332

Mauchly, John W. (1946), *Digital and Analogy Computing Machines, Lecture 1.* In: Campbell-Kelly, Martin; Williams, Michael R. (eds.), The Moore School Lectures. Theory and Techniques for Design of Electronic Digital Computers (Charles Babbage Institute, Reprint Series for the History of Computing, Bd. 9), London, Los Angeles, San Francisco 1985, 25–40

Mauchly, John W. (1947), *Preparation of Problems for EDVAC-Type Machines.* Proceedings of a Symposium on Large Scale Digital Calculating Machinery; nachgedruckt in: Randell, Brian (ed.) (1973), 365–369

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

Meiners., Gustav (1936), *Die Technik selbsttätiger Steuerungen und Anlagen; neuzeitliche schaltungstechnische Mittel und Verfahren, ihre Anwendung auf den Gebieten der Verriegelungen und der selbsttätigen Steuerungen.* München, Berlin

Menabrea, Luigi F. (1842), *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., 666–731; wiedergedruckt in: Morrison, Philip; Morrison, Emely (eds.), Charles Babbage and His Calculating Engines. Selected Papers by Charles Babbage and Others, New York 1961; Internet: http://psychclassics.yorku.ca/Lovelace/menabrea.htm

Merrifield, C. W. (1879), *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, 92–102, London 1879; wiedergedruckt in: Randell, Brian (ed.) (1973), 53–63

Nake, Frieder (2003), *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, 339–365

Neumann, John von (1945), *First Draft of a Report on the EDVAC.* Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia 30. Juni 1945; wiedergedruckt in: Annals of the History of Computing 15 (1993) 4, 27-67

Neumann, John von (1963), *Design of Computers, Theory of Automata and Numerical Analysis.* In: Taub, Abraham H. (ed.), Complete Works of John von Neumann, 6 Bde. Oxford (UK), New York 1961-63, Bd 5

Oxford English Dictionary (1989), *Artikel: program, programme, programmer, programming.* 2nd ed., vol. XII, Oxford, 589–592

Pflüger, Jörg-Martin (1994), *Über die Verschiedenheit des maschinellen Sprachbaues.* In: Bolz, Norbert; Kittler, Friedrich; Tholen, Christoph (Hrsg.), Computer als Medium, München, 161–181

Pflüger, Jörg-Martin (2002), *Language in Computing.* In: Doerries, Matthias (ed.), Experimenting in Tongues: Studies in Science and Language, Stanford

Pflüger, Jörg-Martin (2003a), *Writing, Building, Growing: Leitvorstellungen der Programmiergeschichte.* In: Hellige, Hans Dieter, (Hrsg.), Geschichten der Informatik. Visionen, Paradigmen und Leitmotive, Berlin, Heidelberg, New York, 275–320

Pflüger, Jörg-Martin (2003b), *Konversation, Manipulation, Delegation. Zur Ideengeschichte der Interaktivität.* In: Hellige, Hans Dieter, (Hrsg.), Geschichten der Informatik. Visionen, Paradigmen und Leitmotive, Berlin, Heidelberg, New York, 367–408

Randell, Brian (ed.) (1973), *The Origins of Digital Computers. Selected Papers.* Berlin, Heidelberg, New York

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

Sammet, Jean E. (1969), *Programming Languages: History and Fundamentals.* Englewood Cliffs, N. J.

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

Schmid, Wolfgang (1941), *Untersuchung der Arbeitsspiele der verschiedenen selbsttätigen Steuerungen im Fertigungswesen.* Feinmechanik und Präzision 49 (6), 65–69

Schmid, Wolfgang; Olk, Friedrich (1939), *Fühlergesteuerte Maschinen.* Essen

Stach, Heike (1998), *Beschreiben, konstruieren, programmieren. Zur Verschmelzung von Theorie und Gegenstand.* In: Siefkes, Dirk; Eulenhöfer, Peter; Stach, Heike; Städtler, Klaus

(Hrsg.), Sozialgeschichte der Informatik. Kulturelle Praktiken und Orientierungen, Wiesbaden, 213–229

Stibitz, George R. (1947), *The Organization of Large Scale Calculating Machinery*. In: Harvard University Computation Laboratory (ed.), 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, MA 1948; wiedergedruckt in: Charles Babbage Institute (ed.), Proceedings of a Symposium on Large Scale Calculating Machinery. Reprint Series for the History of Computing Bd. 7 , London, Los Angeles, San Francisco, 91-100

Strauch, Helmar (1937), *Selbsttätige Steuerung mechanischer Bewegungen durch Lochkarten.* Maschinenbau 5 (9), 476–478

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

Torres y Quevedo, Leonardo (1914), *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, 601-611; übersetzt in: Essays on Automatics. Its Definition - Theoretical Extent of Its Applications (1914). In: Randell, Brian (ed.) (1973), 87–105

Torres y Quevedo, Leonardo (1920), *Arithmomètre electromécanique.* In Bulletin de la société d'encouragement pour l'industrie nationale, Bd. 119, 588-599; englische Übersetzung: Electro-mechanical Calculating Machine. In: Randell, Brian (ed.), (1973), 107–118

Turing, Alan M. (1945), *Proposal for Development in the Mathematics Division of an Automatic Computing Engine (ACE).* presented to the National Physical Laboratory,; wiedergedruckt in: Computer Science 57, National Physical Laboratory, Teddington 1972; wiedergedruckt in: Carpenter, B. E.; Doran, R. W. (eds.), 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, 20-105

Turing, Alan M. (1947), *Lecture to the London Mathematical Society on 20 February 1947.* wiedergedruckt in: Carpenter, B. E.; Doran, R. W. (eds.), 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, 106–124

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

Weinhart, Karl (Hrsg.) (1990), *Informatik und Automatik. Führer durch die Ausstellung.* Deutsches Museum, München

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

Zemanek, Heinz (1988), *Hollerith und Schäffler: Zwei Pioniere der Lochkartentechnik. Datenverarbeitung am Ende des 19. Jahrhunderts.* In: Ausgewählte Beiträge zu Geschichte und Philosophie der Informationsverarbeitung (Schriftenreihe der Österreichischen Computer Gesellschaft, Bd. 43), Wien, München, 13–35

Zuse, Horst (2003), *Konrad Zuses Visionen und Konzepte für die Anwendung seiner Rechenmaschinen.* In: Hellige, Hans Dieter, (Hrsg.), Geschichten der Informatik. Visionen, Paradigmen und Leitmotive, Berlin, Heidelberg, New York, 61–77

Zuse, Konrad (1936), *Die Rechenmaschine des Ingenieurs*, *Abschnitt: Die Aufstellung der Rechenpläne.* Nachlasskopien-Bestand des Heinz-NixdorfForums, Paderborn, HNF 009/001; Zuse-Internet-Archiv; ZIA 0234

Zuse, Konrad (1937), *Paralleloperationen bei Programmen. Tagebuchnotiz vom 16.7.1937. Gedanken zum künstlichen Gehirn*. HNF 025/015; ZIA 0419

Zuse, Konrad (1938), *Programmspeicherung. Tagebuchnotizen über die Entwicklung von starren Rechenplänen und lebenden Rechenplänen*. HNF 025/011; ZIA 0417

Zuse, Konrad (1939), *Plankalkül Vorarbeiten. Tagebuchnotiz vom 25.5.1939 über flexible Angabenstrukturen. Mechanisches Gehirn.* HNF 025/012; ZIA 0418

Zuse, Konrad (1945), *Plankalkül (Fassung von 1945)*. Abschrift 1946, HNF 011/008; ZIA 0233

Zuse, Konrad (1946), *Zuse-Rechengeräte*. Ms. Mai 1946, HNF 010/005; ZIA 0332

Zuse, Konrad (1947), *Zuse-Rechengeräte.* Zuse-Ingenieurbüro Hopferau (Hrsg.), HNF 011/008; ZIA 0743