Von ›bits‹ zu ›words‹

John von Neumanns *linguistic turn* als Ursprung digitaler Schriftlichkeit

Gabriele Gramelsberger

1. Einleitung

Digitale Schriftlichkeit als Programmieren, Prozessieren und Codieren von Schrift setzt bereits Schrift in zweifacher Weise voraus. Zum einen als materialisierte Schrift der Mechanisierung und Elektrifizierung der Sprache, die als logisch-elektronische Grundlage das Digitale erst ermöglicht; zum anderen als Maschinencode. Es ist die Idee des Maschinencodes, die John von Neumann als *linguistic turn* in die Instruierung elektrischer Computer Mitte der 1940er Jahre einführte und die das gesamte Potential des Phänomens der digitalen Schriftlichkeit überhaupt erst ermöglicht. Aus diesem *linguistic turn*, der von John von Neumann später kurzerhand »coding« genannt wurde, entwickelten sich ab Mitte der 1950er Jahre über erste Codierhilfen wie Assembler oder Compiler die höheren Programmiersprachen,[1] die nach über achttausend Jahren Schriftentwicklung einen neuen Schrifttyp darstellen.[2] Alle Phänomene des Digitalen – von der Automatisierung und Datafizierung bis hin zu

1 Herman H. Goldstine/John von Neumann: Planning and Coding Problems for an Electronic Computing Instrument, 1947, Part II, Vol 1. In: John von Neumann (Hg.): *Collected Works*, Bd. 5: *Design of Computers, Theory of Automata and Numerical Analysis*. Oxford: Pergamon Press 1963, 80–151, 103.

2 Achttausend Jahre Schriftentwicklung unter Berücksichtigung der in China gefundenen Jiahu-Zeichen (6.600 v. Chr.) und der Donauschrift (5.500 v. Chr.).

maschinellen Lernverfahren und Künstlicher Intelligenz – haben ihren Ursprung in diesem *linguistic turn*.

Die Bezeichnung *linguistic turn* im Zusammenhang mit John von Neumanns Einführung von Code und Coding ist eine retrospektive Zuschreibung der Computerhistoriker Mark Priestley und Thomas Haigh. In einem Beitrag über frühe US-amerikanische Computer stellten sie fest: »Computer programming took a linguistic turn, largely due to the fact of John von Neumann during the famous ›First Draft of a Report on the EDVAC‹ came up with the first thing that we would recognize as an order code, a machine programming linguistic coded instruction.«[3] EDVAC [Electronic Discrete Variable Automatic Computer], dessen Design von Neumann 1945 beschrieben hatte, wurde zum Vorbild aller modernen Digitalcomputer. Mit EDVAC wurden Programme erstmals als Daten behandelt, binär codiert und im internen Speicher verarbeitet. Dieses Design wird bis heute von-Neumann-Architektur genannt. Den Bericht über das Design von EDVAC leitete von Neumann wie folgt ein:

> »The considerations which follow deal with the structure of a very high speed automatic digital computing system, and in particular with its logical control. […] These instructions must be given in some form which the device can sense: Punched into a system of punch-cards or on teletype tape, […]. All these procedures require the use of some code, to express the logical and the algebraical definition of the problem under consideration, as well as the necessary numerical material.«[4]

Damit war die Bühne von Hardware und Software eröffnet, auf welcher sich von da an – theatral formuliert – die Dramen des Digitalen abspielen sollten. Vor dem Hintergrund der Wirkmächtigkeit dieses *linguistic*

3 Mark Priestley/Thomas Haigh: Working on ENIAC: The Lost Labors of the Information Age. In: *MITH Digital Dialogues on opentranscrips.org*, 02.18.2016. http://opentranscripts.org/sources/mith-digital-dialogues/ (zuletzt abgerufen am 09.06.2023).

4 John von Neumann: *First Draft of a Report on the EDVAC*. University of Pennsylvania: Moore School of Electrical Engineering 1945, 1.

turns ist die Analyse der verschiedenen Arten des Schriftgebrauchs für ein kritisches Verständnis der digitalen Schriftlichkeit von Interesse.

2. Mechanisierung der Sprache

Der Digitalcomputer, das wird gern vergessen, ist der materialisierte Endpunkt einer jahrhundertealten Entwicklung der Mechanisierung und Elektrifizierung der Sprache.[5] Zu Beginn dieser Entwicklung stand das philosophische Programm der Operationalisierung des Geistes. Rationalisten wie René Descartes und Gottfried Wilhelm Leibniz als auch Empiristen wie John Locke und David Hume widmeten sich der Analyse und Operationalisierung der Erkenntnisfunktionen des Subjekts. Von Descartes' analytischem Problemlösungsverfahren über Lockes Definition von Wissen als »die Auffassung der Verbindung und Übereinstimmung oder der Nichtübereinstimmung und des Widerstreits unserer einzelnen Vorstellungen […]. Darin allein besteht es«[6] bis hin zu Leibniz' Ersetzung von Ähnlichkeit durch Äquivalenz (*salva veritate*) formierte sich die konzeptuelle Grundlage des Digitalen bereits in der Neuzeit. Insbesondere Leibniz hatte nicht nur ein Faible für Logik und Kalküle und erfand das binäre Rechnen, sondern er war sich schon 1693 über die Tragweite einer an Maschinen delegierten Sprache in Form von Kalkülen bewusst. Sie wäre zukünftig eine »Ergänzung der sinnlichen Anschauung und gleichsam ihre Vollendung« und würde auch »in den Beschreibungen der Mechanismen der Natur« äußerst nützlich sein.[7]

Was mit den Analysen der Rationalisten und Empiristen entstand, war ein formales Verständnis von Verstandesfunktionen, das von allem Inhaltlichen abstrahierend die Verstandesfunktionen als Operationen

5 Gabriele Gramelsberger: *Philosophie des Digitalen. Zur Einführung*. Hamburg: Junius 2023.

6 John Locke: *Versuch über den menschlichen Verstand*. Berlin: Heimann 1872, 4. Buch, Kap. 1, §2.

7 Gottfried Wilhelm Leibniz: De analysi situs/Kalkül der Lage. In: Ders.: *Philosophische Werke*, 1. Bd. Hamburg: Meiner 1996, 69–76, 76.

fasste. Bei aller Differenz gelangen beide philosophische Strömungen am Ende zu sehr ähnlichen Operationsmodellen des Denkens. So waren sie sich einig, dass Verstandesfunktionen in sehr einfachen Operationen des Vergleichens, Feststellens von Identität oder Nicht-Identität, des Auseinanderdividierens (Analyse) und des Zusammensetzens (Synthese) bestanden. Das Feststellen von Identität oder Nicht-Identität (Negation) sowie das Zusammensetzen in Form der Konjunktion und Disjunktion konstituieren das Spektrum der aussagenlogischen Verknüpfungen, die heute die primitiv-rekursive und entscheidungsdefinite Grundlage des Digitalen auf Basis der Booleschen Algebra (AND, OR, NOT) bilden. Hinzukommt, dass mit Leibniz' Prinzip *salva veritate* sich eine Aussage durch eine andere Aussage formal ersetzen lässt, wenn dadurch der Wahrheitswert der Aussage oder des Satzes nicht verändert wird.

Doch auch wenn sich die Verstandesfunktionen als Operationen bereits im 16. und 17. Jahrhundert abstrahieren ließen, so fehlte es an einem adäquaten Ausdrucksmittel. Denn seit der Antike nutzte die Philosophie die aristotelische Logik (Syllogistik), die als eine Sprachlogik auf semantische Inhalte konzentriert war. Erst George Boole gelang es in seinem epochalen Werk *The Mathematical Analysis of Logic* von 1847, ein adäquates Ausdrucksmittel zu entwickeln. Boole hatte nichts geringeres als die Formalisierung der Logik selbst zum Ziel. Dazu analysierte er die Struktur der Urteilsformen der aristotelischen Syllogistik, um zu seinem »Calculus of Logic« zu gelangen; »a method resting upon the employment of Symbols, whose laws of combination are known and general, and whose results admit of a consistent interpretation.«[8] Die Gleichsetzung von Logik und Zeichengebrauch, die heute symbolische Logik genannt wird, erforderte aber die Einführung einer neuen Operation: eines formalen Mechanismus der Klassifikation und der Auswahl von Individuen. Klassenbildung respektive Selektion, so Boole, sollten nicht mehr länger ein Geschäft der aristotelischen Metaphysik, sondern der Mathematik sein. Erst dieser Selektionsmechanismus ermöglichte es Boole, die Struktur

8 George Boole: *The mathematical analysis of logic, being an essay towards a calculus of deductive reasoning*. Cambridge: Macmillan, Barclay, & Macmillan u.a. 1847, 4.

der aristotelischen Logik zu formalisieren sowie die logischen Urteilsformen als mathematische Gleichungen darzustellen.[9] Aus den aristotelischen Urteilsformen der allgemein bejahenden oder verneinenden Urteile sowie der partikular bejahenden oder verneinenden Urteile wurden Formeln: Aus »Alle X sind Y« wurde bei Boole »xy = x«, aus »Kein X ist Y« wurde »xy = 0«, aus »Einige X sind Y« wurde »v = xy« und aus »Einige X sind nicht Y« wurde »v = x(1-y)«. Mit diesen Formeln ließ sich nun rechnen. Auf dieser Basis rekonstruierte Boole die aristotelische Logik als formale Aussagenlogik, die später axiomatisiert zur booleschen Algebra wurde, die bis heute die logische Grundlage der Schaltungen von Digitalcomputern bildet. Die boolesche Algebra ist damit der zentrale Markstein der Mechanisierung der Sprache. Allerdings, und darauf hatte etwas später Charles S. Peirce aufmerksam gemacht, formalisierte Booles Logikkalkül nur einen Bruchteil logischer Schlussformen der aristotelischen Syllogistik. Daher erweitere Peirce später Booles Kalkül zu einem Relationskalkül und entwarf damit eine erste Prädikatenlogik.[10]

3. Elektrifizierung der Sprache

Doch Mechanisierung der Sprache alleine genügt nicht, um diese an Maschinen wie den Digitalcomputer zu delegieren. Hier kommt ein weiterer Schriftgebrauch ins Spiel, der die Elektrifizierung der Sprache ermöglichte und ein Jahrhundert nach Boole von Claude Shannon ins Spiel gebracht wurde. Im Kontext der Telefonie entwickelte sich zu Beginn des 20. Jahrhunderts die Theorie der gepulsten Signalübertragung von

9 Was die Logik hierfür prädestinierte, war aus Booles' Perspektive Folgendes: »That which renders Logic possible, is the existence in our minds of general notions, – our ability to conceive of a class, and to designate its individual members by a common name. The theory of Logic is thus intimately connected with that of Language.« Boole: *Mathematical Analysis of Logic*, 4–5.

10 Charles S. Peirce: Description of a Notation for the Logic of Relatives, Resulting from an Amplification of the Conceptions of Boole's Calculus of Logic. In: *Memoirs of the American Academy of Sciences* 9 (1870), 317–378.

Sprache. Die Frage, die es dabei zu lösen galt, war, welche diskrete Übertragungsrate die effizienteste für ein analoges Sprachsignal wäre, ohne Informationen der Sprachübertragung zu verlieren. Die Bell Lab Forscher Harry Nyquist und Ralph Hartley entwickelten eine Theorie der Abtastung und Quantisierung (Quantisierungsstufen) kontinuierlicher Signale für die elektrische Kommunikation. Um den Übergang vom physikalischen Phänomen elektromagnetischer Trägerwellen zum Symbol zu fassen, bedarf es des Wechsels von Sinuswellen zu Logarithmen sowie der Zuordnung von reellen Zahlen (*digits*) zu den einzelnen Quantisierungsstufen. Auf diese Weise lässt sich ein kontinuierliches Signal in ein Digitalsignal transformieren, welches jeder Quantisierungsstufe einen eindeutigen (reellen) Zahlenwert zuordnet.

Im Sinne einer effizienten Übertragung von Informationen führte Shannon 1948 einen statistisch-stochastischen Ansatz in die elektrische Informationsübertragung ein, indem er die Informationsentropie (H) als Maßeinheit für den Informationsgehalt einer Symbolfolge definierte. Dabei betrachtete Shannon den Informationsgehalt einer Symbolfolge rein statistisch als Häufigkeitsvorkommnis von Sprachzeichen und Zeichenkombinationen. In natürlichen Sprachen wie dem Englischen ist folgendes offensichtlich: »The letter E occurs more frequently than Q, the sequence TH more frequently than XP etc.«[11] Der entscheidende Kunstgriff, den Shannon anwandte, war aber folgender: Er stellte Informationen mit Markov-Ketten dar, um die Wahrscheinlichkeit des Vorkommens eines Symbols in einer Symbolreihe sowie die Wahrscheinlichkeit der Abhängigkeit des Vorkommens eines Symbols von den vorhergehenden vorherzusagen. Je komplexer das Vorhersagemodell (Markov-Kette), desto mehr nähert sich die Vorhersage den tatsächlichen Symbolvorkommnissen an. Auf dieser Basis kann Shannon nun die Entropie einer Symbolfolge (Nachricht) bestimmen, um so die notwendigen Übertragungskapazitäten in Form von Bits pro Sekunde (*b*inary digi*t*) zu berechnen. Eine hohe Entropie bedeutet eine hohe Redundanz, also statistische

11 Claude Shannon: A Mathematical Theory of Communication. In: *Bell System Technical Journal* 27 (1948), 379–423 und 623–656, 385.

Regelmäßigkeit, in einer Symbolfolge. Ein Werk wie James Joyces *Finnegans Wake* hat eine niedrige Entropie im Unterschied zu *Basic English*, das nur etwa 850 Wörter umfasst. Je redundanter eine Nachricht ist, desto geringer muss die Kapazität für eine verlustfreie Datenübertragung sein und desto weniger Bits werden für die Codierung einer Nachricht benötigt. Eine Nachricht, bestehend aus zwei gleichwahrscheinlichen Zeichen, hat nach Shannon den Informationsgehalt $\log2(\frac{1}{2})$ und benötigt 1 Bit zur Übertragung. Ob ein Bit (Binärzahl) mit den Ziffern 0 und 1 dargestellt wird, hängt von der gewählten Konvention des Alphabets ab.

Shannons Wechsel von der Analyse zur Vorhersage erforderte die Kenntnis über die typischen Wahrscheinlichkeiten des Vorkommens von Symbolen in einer Sprache. Andrej A. Markov hatte dies für die Buchstabensequenzen in der russischen Literatur Ende des 19. Jahrhunderts untersucht.[12] Friedrich W. Kaeding hatte die Häufigkeit von Buchstaben, Silben und Wörtern mit einem Team von über eintausend Freiwilligen für die deutsche Sprache analysiert und 1898 publiziert.[13] Am Ende hatte Kaeding ein Korpus von über 10 Millionen Wörter und 60 Millionen Buchstaben zusammengetragen; ein Korpus, das in einem derartigen Umfang erst wieder ab den 1970er Jahren mit Hilfe von Computern generierbar wurde. Diese quantitativ-statistischen Analysen der Sprache waren nicht nur Vorläufer der heutigen Digital Humanities, sondern ermöglichten erst die Verbindung von elektrischer Energie und Symbol.

Beide Arten des (Schrift-)Sprachgebrauchs, der logische und der statistisch-stochastische, verbinden sich in Flip-Flop-Schaltungen

12 Andrej A. Markoff: *Wahrscheinlichkeitsrechnung*. Übers. v. Heinrich Liebmann. Leipzig: Teubner 1912; Philipp von Hilgers/Vladimir Velminski (Hg.): *Andrej A. Markov. Berechenbare Künste*. Zürich/Berlin: diaphanes 2007.

13 Friedrich W. Kaeding: *Häufigkeitswörterbuch der deutschen Sprache: Festgestellt durch einen Arbeitsausschuss der deutschen Stenographiesysteme*. Steglitz bei Berlin: Selbstverlag des Herausgebers/E.S. Mittler & Sohn 1898; Toni Bernhart: Von Aalschwanzspekulanten bis Abendrotlicht. Buchstäbliche Materialität und Pathos im Häufigkeitswörterbuch der deutschen Sprache von Friedrich Wilhelm Kaeding. In: Ralf Klausnitzer/Carlos Spoerhase/Dirk Werle (Hg.): *Ethos und Pathos der Geisteswissenschaften*. Berlin/Boston: De Gruyter 2015, 165–190.

(*on/off*) als Grundeinheiten der elektrifizierten Information, insofern, wie Shannon schrieb, »a relay or a flip-flop circuit can store one bit of information.«[14] Flip-Flop-Schaltungen sind aus mehreren Logikgattern zusammengesetzte Schaltungen. Logikgatter wiederum sind Anordnungen von Schaltungen zur (binären) Realisierung boolescher Funktionen gemäß Shannons Schaltalgebra von 1938, die die logischen Operatoren AND, OR, XOR sowie deren Negationen repräsentieren.[15] Aus diesen Schaltungen lässt sich die gesamte arithmetische und logische Funktionalität digitaler Computer wie auch ihre Datenprozessierungs- und -speicherkapazitäten konstruieren.

4. John von Neumanns *linguistic turn*

Digitale Schriftlichkeit gründet in diesen beiden, hoch technisierten Arten des (Schrift-)Sprachgebrauchs, die jedoch am Ende nur Signale oder Bits übriglassen und daher einer mnemotechnischen Erschließung bedürfen. Eine solche mnemotechnische Erschließung ist notwendig, um den Vorteil der freien Programmierbarkeit von Computern als allgemeinen Maschinen nutzen zu können. Ohne diese Erschließung ist die Eingabe eines Programms mühsam, denn für jede neue Aufgabenstellung mussten die zahlreichen Schaltungen der ersten elektrischen Computer einzeln per Hand verbunden werden. Beispielsweise mussten die 17.468 Elektronenröhren, 7.200 Dioden, 1.500 Relais, 70.000 Widerstände und 10.000 Kondensatoren der Rechen-, Zähl- und Speichereinheiten von ENIAC (Electronic Numerical Integrator and Computer), der Vorläufer von EDVAC, für jede neue Berechnung neu verkabelt werden. »Setting up the ENIAC meant plugging and unplugging a maze of cables and setting arrays of switches. In effect, the machine had to be rebuilt for each

14 Shannon: A Mathematical Theory of Communication, 379.

15 Claude Shannon: A Symbolic Analysis of Relay and Switching Circuits. In: *Transactions of the American Institute of Electrical Engineers* 57 (1938), 38–80.

new problem it was to solve.«[16] Während das Verkabeln (Programmeingabe) per Hand Tage dauerte, konnte ENIAC, basierend auf Daten, die mit Lochkarten eingegeben wurden, komplexe Berechnungen in Stunden oder sogar Minuten lösen. Doch auch die Zerlegung von Berechnungen per Hand in Teilberechnungen und schaltungsgerechte Abläufe sowie die Übersetzung in einen Plan der Verkabelung, dauerten ihre Zeit, ebenso wie die Fehlersuche und -korrektur. ENIAC, so wird kolportiert, rechnete nur rund zwei Stunden pro Woche. Der Rest der Zeit musste in die Verkabelung wie auch Fehlersuche investiert werden.

Eben diese Situation fand John von Neumann vor, als er sich als Mathematiker mit dem Bau aber auch dem Instruieren der ersten elektronischen Computer in den USA beschäftigte. Von Neumann, ein ungarischer Mathematiker, der bei David Hilbert in Göttingen studiert und sich in Berlin habilitiert hatte, emigrierte 1929 in die USA. Er war während des Zweiten Weltkrieges in Los Alamos im Manhattan-Projekt für die Berechnung der Ausbreitung von Explosionswellen zuständig. Allerdings erwiesen sich die Ausbreitungsmodelle basierend auf partiellen Differentialgleichungen als zu komplex, als dass sie per Hand analytisch lösbar waren. Der Mathematiker Stanislav Ulam beschrieb die Situation 1943 in Los Alamos wie folgt:

> »The blackboard was filled with very complicated equations that you could encounter in other forms in other offices. […] looking at these I felt that I should never be able to contribute even an epsilon to the solution of any of them. But during the following days, to my relief, I saw that the same equations remained on the blackboard. I noticed that one did not have to produce immediate solutions.«[17]

Vor diesem Hintergrund erkannte von Neumann nicht nur die Bedeutung freiprogrammierbarer Digitalrechner, sondern entwickelte Metho-

16 Paul E. Ceruzzi: *A History of Modern Computing*. Cambridge, MA: The MIT Press 1998, 21.

17 Stanislaw Ulam: Von Neumann: The Interaction of Mathematics and Computing. In: Nicholas Metropolis/Jack Howlett/Gian-Carlo Rotta (Hg.): *A History of Computing in the Twentieth Century*. New York: Academic Press 1980, 93–99, 95.

den zur Diskretisierung von Differentialgleichungsmodellen, um diese numerisch zu berechnen respektive die Berechnungen an Computer delegieren zu können. Heute nennen wir diese Methode numerische Simulation partieller Differentialgleichungsmodelle, beispielsweise von Wetter- oder Klimamodellen – oder kurz: Computersimulation.[18]

Die Mühsamkeit der Verkabelung von ENIAC zur Berechnung kurzer Programme einerseits sowie die Notwendigkeit andererseits, immer umfangreichere Berechnungen für Differentialgleichungsmodelle durchzuführen, waren der Grund, warum von Neumann über »a very high speed automatic digital computing system, and […] its logical control« nachdachte.[19] Er entwarf nicht nur die bis heute verwendete Architektur von Digitalcomputern, die Programme als Daten behandelte und über Lochkarten einlesbar oder sogar intern speicherbar machte, sondern vollzog für seine neue Maschine einen *linguitsic turn*.[20] Konkret meint dies die Verwendung von Codeworten für die logische Kontrolle der Maschine. »It is therefore our immediate task to provide a list of the orders which control the device, i.e. to describe the code used in the device, and to define the mathematical and logical meaning and the operational significance of its code words.«[21] Von Neumann gibt am Ende seines Designs von EDVAC eine Liste von Codeworten und deren Bedeutung. Beispielsweise meint das Codewort »i_l=1« in seiner Kurzform »whup« den Befehl »to carry out the operation w in CA and to

18 Der Begriff Computersimulationen reicht über die numerische Simulation partieller Differentialgleichungen (deterministische Simulation) hinaus und bezeichnet weitere Simulationsformen wie stochastische Simulationen (z.B. Monte-Carlo-Simulation). Gabriele Gramelsberger (Hg.): *From Science to Computational Sciences. Studies in the History of Computing and its Influence on Today's Sciences*. Zürich/Berlin: diaphanes 2011.

19 von Neumann: *Report on the EDVAC*, 1. Von Neumann konzipierte Computer als Maschinen mit Rechenwerk (arithmetisch-logische Einheit), Steuerwerk, Busseinheit (Daten- und Energieübertragung zwischen den Komponenten), Speicherwerk sowie Eingabe-/Ausgabewerk.

20 Thomas Haigh/Mark Priestley/Crispin Rope: *ENIAC in Action: Making and Remaking the Modern Computer*. The MIT Press: Cambridge, MA 2016.

21 von Neumann: *Report on the EDVAC*, 85. Unterstreichungen im Originaltext.

dispose of the result. […] h means that the result is to be held in O_{ca}. up means, that the result is to be transferred into the minor cycle ρ in the minor circle u.«[22]

Es ist die symbolische Ersetzung X (*words*) für die Maschinenoperation Y, die den *linguistic turn* markiert. Was heute selbstverständlich anmutet, war 1945 neu und revolutionierte die Computerentwicklung. Denn diese symbolische Ersetzung (Maschinencode für Maschinenoperationen) vereinfachte nicht nur die Eingabe eines Programms in einen Computer, sondern stellte eine erste Form der Automatisierung dar und zwar dessen, was bis dahin per Hand zur Verkabelung einer Maschine ausgeführt werden musste. 1951 ging Maurice Wilkes, der an der Entwicklung des Manchester Computers beteiligt war, einen entscheidenden Schritt weiter. Er schlug vor, basale Schalt-, Rechen- und Steuerabläufe unterhalb der Ausführungsebene von Maschinencodes festzulegen, und nannte diese unveränderbare Form des Codes »micro-operations« (Microcode): »Each true machine operation is thus made up of a sequence or ›micro-program‹ of micro-operations. […] Only 40 micro-orders are required to perform all these operations [add, subtract, multiply, transfer etc.].«[23] Microcode wird in der Regel fest in einen Computer respektive Prozessor integriert. Maschinencodes adressieren dann die Microcodes, die lediglich Abfolgen von Nullen und Einsen sind.

Die mnemotechnische Revolution des Maschinencodes initiierte eine Kaskade an Folgephänomenen, in deren Verlauf sich die symbolische Ersetzung immer mehr an den menschlichen Sprachgebrauch annäherte. Compiler waren erste Programme, die Programmcode in Maschinencode übersetzten, die dann in einem Computer wiederum via Microcodes in Maschinenoperationen transferiert wurden. Allerdings wurden diese ersten Programmierhilfen nicht von allen begeistert aufgenommen. »At that time [1954], most programmers wrote symbolic machine instructions exclusively […] they firmly believed that any mechanical coding method would fail to apply the versatile ingenuity

22 von Neumann: *Report on the EDVAC*, 100.

23 Maurice V. Wilkes: The Best Way to Design an Automated Calculating Machine. In: *Manchester University Computer Inaugural Conference* (1951), 182–184, 183–184.

which each programmer felt he possessed and constantly needed in his work.«[24] Diese ersten Programmierhilfen entwickelten sich Mitte der 1950er Jahre zu Programmiersprachen, die einerseits weitere Vereinfachungen des Programmierens zum Ziel hatten (Automatisierung), andererseits aus der Kombination einfacher Operationen zunehmend komplexere Programmierbefehle ermöglichten. Dadurch konnte das Möglichkeitspotential des frei programmierbaren Digitalcomputers immer besser ausgeschöpft werden. John Backus beispielsweise, der die höhere Programmiersprache FORTRAN (Formula Translation) entwickelte und 1954 vorstellte, stellte sich folgende Frage:

> »What could be done now to ease the programmer's job? Once asked, the answer to this question had to be: Let him use mathematical notations. But behind that answer […] there was the really new and hard question: Can a machine translate a sufficiently rich mathematical language into a sufficiently economical machine program to make the whole affair feasible?«[25]

Backus' letzte Frage ist nicht trivial, denn es muss sichergestellt sein, dass der Computer tatsächlich die programmierte Berechnung ausführt. Solange die Programminstrukteure dies direkt in Maschinencode übertragen, kann er oder sie dies selbst überprüfen. Wird die Übersetzung jedoch durch eine Programmiersprache via einen Compiler oder einen Interpreter automatisiert, muss man sich auf diese Programmierhilfen verlassen. Die Entwicklung von Programmierhilfen und Programmiersprachen stellte die erste Welle der Automatisierung durch das Digitale dar. Viele weitere sollten folgen bis hin zum aktuellen Hype der Künstlichen Intelligenz.

---

24 John Backus/William P. Heising: FORTRAN. In: *IEEE Transactions on Electronic Computing* 13 (1964), 382–385, 382.

25 John Backus: Programming in America in the 1950s. In: Nicholas Metropolis/Jack Howlett/Gian-Carlo Rotta (Hg.): *A History of Computing in the Twentieth Century*. New York: Academic Press 1980, 125–135, 131.

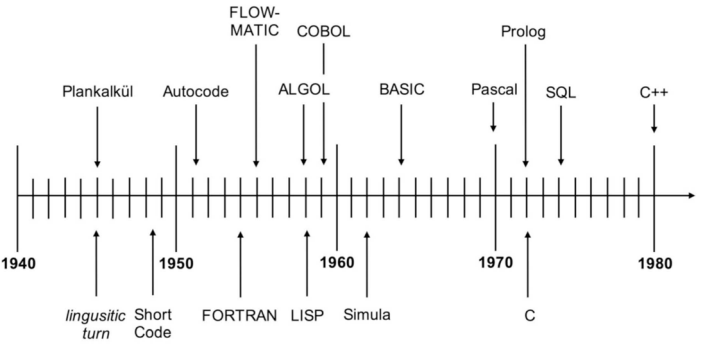
5. Protodigitale Schriftlichkeit

Mit dem Übergang von Maschinenoperationen und Microcode zu Maschinencode und später zur Automatisierung durch die zunehmend komplexeren Programmcodes der Compiler und höheren Programmiersprachen setzte sich nach achttausend Jahren Schriftentwicklung ein neuer Typ von Schriftgebrauch durch: autooperative Schriften, die einem Zeichengebrauch folgen, »den man den deklarativen Zeichengebrauch nennen könnte, bei dem ein Autor eine Aufgabe anschreibt (deklariert), die dann von einer Maschine exekutiert wird.«[26] Digitale Schriftlichkeit basiert auf diesem deklarativen Zeichengebrauch, der wiederum in dem skizzierten Programm der Mechanisierung und Elektrifizierung der Sprache gründet. Doch digitale Schriftlichkeit verlässt sich auch auf den skizzierten *linguistic turn*, insbesondere auf dessen erste Errungenschaft: Codeworte, Microcodes und schließlich Maschinencodes, die sich als Formen einer protodigitalen Schriftlichkeit charakterisieren lassen. Das Adjektiv ›protodigital‹ bezieht sich darauf, dass Codeworte, Microcodes und Maschinencodes eine notwendige Vorform der digitalen Schriftlichkeit sind, gleichwohl es sich um digitale Anweisungen handelt (Folgen von Bytes, die sowohl Daten als auch Befehle repräsentieren). Deutlich wird dies auch durch das Adjektiv ›höhere‹, das das Protodigitale insofern in sich trägt, als Programmiersprachen nicht unmittelbar von Computerprozessoren verstanden und ausgeführt werden können, sondern interpretiert oder kompiliert werden müssen. Diese Differenz im Schriftgebrauch hat zur Folge, dass erst höhere Programmiersprachen die gesamte Vielfalt an Möglichkeiten der digitalen Operativität eröffnen, aber zum Preis der Spezialisierung auf Teilbereiche wie die mathematische Operativität (FORTRAN), die logische Operativität (Prolog), die Operativität der Datenverwaltung (SQL) und viele weitere Formen der Operativität. Die maschinischen Grundoperationen respektive Microcodes fungieren

26 Gernot Grube: Autooperative Schrift. In: Ders./Werner Kogge/Sybille Krämer (Hg.): *Kulturtechnik Schrift. Die Graphé zwischen Bild und Maschine*. München: Fink 2005, 81–114, 82.

dabei wie ein Baukasten, mit dem sich durch geschickte Kombinationen zunehmend komplexere, aber eben auch spezialisiertere Programmierbefehle kombinieren lassen. Wenn also von digitaler Schriftlichkeit die Rede ist, sind einerseits diese maschinisch zugerichteten Schriftvoraussetzungen, aber anderseits auch die Ambiguität einer zunehmenden Erweiterung digitaler Operativität bei gleichzeitiger Spezialisierung mitzudenken.

Abb. 1: Entwicklung von frühen Programmierhilfen und -sprachen[27]



Doch bereits in den 1960er und 1970er Jahren zeigte sich, dass für von Neumanns wegweisende Konzepte, sowohl die des Computers wie

27 Parallel zu John von Neumann hatte sich der Berliner Konrad Zuse zwischen 1942 und 1945 Notizen zu einer höheren Programmiersprache (Plankalkül) für seinen geplanten Computer Z3 gemacht, konnte diesen jedoch aufgrund des Krieges nicht verwirklichen. Der Plankalkül wurde erst 1972 vollständig veröffentlicht und ist Anfang der 2000er Jahre erstmals rekonstruiert und implementiert worden. Konrad Zuse: Über den allgemeinen Plankalkül als Mittel zur Formulierung schematisch-kombinativer Aufgaben. In: *Archiv der Mathematik* 1 (1948/49), 441–449; Friedrich L. Bauer/Hans Wössner: The »Plankalkül« of Konrad Zuse: A Forerunner of Today's Programming Languages. In: *Communications of the ACM* 15 (1972), 678–685.

seiner mnemotechnischen Revolution (*linguistic turn*), ein hoher Preis zu zahlen war. Backus hat dies 1978 treffend formuliert:

> »In order to understand the problems of conventional programming languages, we must first examine their intellectual parent, the von Neumann computer. […] In its simplest form a von Neumann computer has a central processing unit (or CPU), a store, and a connecting tube that can transmit a single word between the CPU and the store (and send an address to the store). I propose to call this tube the *von Neumann bottleneck*. The task of a program is to change the contents of the store in some major way; when one considers that this task must be accomplished entirely by pumping single words back and forth through the von Neumann bottleneck, the reason for its name becomes clear. Ironically, a large part of the traffic in the bottleneck is not useful data but merely names of data, as well as operations and data used only to compute such names.«[28]

Es ist klar, dass dieser von-Neumann-Flaschenhals die Programmausführung erheblich verlangsamte. Dies wurde und wird zwar durch zunehmend kleinere und schnellere Mikroprozessoren kaschiert. Doch das ist nur ein Aspekt. »More importantly, it is an intellectual bottleneck that has kept us tied to a-word-at-a-time thinking instead of encouraging us to think in terms of the larger conceptual units of the task at hand.«[29] Das Problem wird bis heute durch die Entkopplung von Speicher und Rechenwerken über Hierarchien an Zwischenspeicher praktisch gelöst. Maurice V. Wilkes, der das Konzept der Zwischenspeicher entwickelte, motivierte seine »slave memories« wie folgt:

> »The use is discussed of a fast core memory of, say, 32000 words as a slave to a slower core memory of, say, one million words in such a way that in practical cases the effective access time is nearer that of the fast

28 John Backus: Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs. In: *Communications of the ACM* 21 (1978), 613–641, 615.

29 Backus: Can Programming Be Liberated, 615.

> memory than that of the slow memory. […] By a slave memory I mean one which automatically accumulates to itself words that come from a slower main memory, and keeps them available for subsequent use without it being necessary for the penalty of the main memory access to be incurred again.«[30]

Bis heute arbeitet sich die Entwicklung von Computern wie auch Programmiersprachen am von-Neumann-Flaschenhals in der einen oder anderen Weise ab, auch wenn Prozessoren unglaublich klein (2nm), schnell (ExaFlop/s) und komplex (*system on a chip*) geworden sind und die Datenvolumen unfassbare Ausmaße (Zettabytes) angenommen haben. Doch wie umfangreich die digitale Schriftlichkeit, die uns als sichtbares Phänomen auf der Oberfläche digitaler Geräte im Alltag begleitet, auch immer ist, an der konzeptuellen Grundlage ihrer Programmierung, Prozessierung und Codierung hat sich bis heute nicht viel geändert, solange von-Neumann-Computer verwendet werden. Allerdings macht sich mittlerweile ein Effekt bemerkbar, der im Paradox des *linguistic turn* seinen Ursprung hat. Dieses Paradox resultiert aus dem Umstand, dass der *linguistic turn* Maschinenoperationen mnemotechnisch erschließbar macht und so die Nähe zur Sprache wieder herstellt, allerdings zum Preis eines sehr aufwendigen »a-word-at-a-time thinking«, dass zur Produktion von Unmengen an Daten führt und immer mehr Speicher und Computerleistung erfordert.[31] Diese mnemotechnisch Revolution ermöglichte jedoch erst die digitale Revolution gerade wegen des anthropomorphen (Schrift-)Zugangs zum Digitalen.

---

30 Maurice V. Wilkes: Slave Memories and Dynamic Storage Allocation. In: *IEEE Transactions on Electronic Computers EC-14* (1965), 270–271, 270. Das Konzept der Zwischenspeicher wie auch der Trennung von Programm- und Datenzwischenspeichern hat neben dem von-Neumann-Computer (SISD-Computer, *Single Instruction, Single Data*) zu neuen Computerarchitekturen wie Vektorrechnern (SIMD-Computer, *Single Instruction, Multiple Data*) oder Parallelrechnern (MIMD-Computer, *Multiple Instruction, Multiple Data*) geführt. Michael J. Flynn: Some Computer Organizations and Their Effectiveness. In: *IEEE Transactions on Computers* C-21 (1972), 948–960.

31 Backus: Can Programming Be Liberated, 615.

Doch dieser anthropomorphe Zugang droht zunehmend verloren zu gehen. Denn die Unmengen an ikonoklastischen Daten wie auch die enormen Rechengeschwindgeiten von Billionen von Operationen pro Sekunde unterlaufen schlichtweg die menschlichen Wahrnehmungsfähigkeiten. Hinzu kommt der zunehmende Einsatz von KI (Künstlicher Intelligenz) zur automatisierten Kontrolle und Steuerung des Digitalen. Zusammengenommen erleben wir heute einen Effekt der Deanthropomorphisierung des Digitalen, der uns immer weniger direkten Zugang erlaubt.[32] Im Zuge dieser Deanthropomorphisierung geht auch der mnemotechnische Zugang durch von Neumanns *linguistic turn* verloren. Nicht nur weil Programmierung zunehmend an KI delegiert wird, sondern weil die Milliarden smarter Objekte sich effizienter in binärcodierten Microcodes oder elektrischen Signalen als in symbolischer Maschinensprache unterhalten. Der Umweg über mnemotechnische Hilfen von Codeworten und den *linguistic turn* ist aufgrund der unglaublichen Automatisierungskomplexität und -tiefe des Digitalen und dessen Autooperativität nicht länger notwendig. Sie ist allenfalls eine nostalgische Reminiszenz an den Menschen.

32 Eine ausführliche Diskussion dieser Entwicklung findet sich in Gramelsberger: *Philosophie des Digitalen*.