THOMAS HAIGH

VON-NEUMANN-ARCHITEKTUR, SPEICHERPROGRAMMIERUNG UND MODERNES CODE-PARADIGMA

Drei Leitbilder früher Rechenanlagen

I. Die Erfindung des Computers

Wer hat denn jetzt letzten Endes den Computer erfunden? Stellt man diese Frage auf einer Party, so wird sie jeden seriösen Computerhistoriker erbleichen lassen; er wird eine Entschuldigung murmeln, um dann schnellstmöglich zum Tisch mit den Getränken zu verschwinden. Unser ganzes Schreiben und Denken über die Computer der 1940er Jahre ist mittlerweile ein Versuch, einer exklusiven Antwort auf diese Frage zu entkommen. Stattdessen verleihen wir jeder alten Maschine und deren Erfinder(n) einen virtuellen Preis, auf dem Sätze eingraviert sind wie «Erster automatischer elektronischer Allzweck-Computer». Solche Preise schmücken die metaphorischen Kaminsimse von John Atanasoff, Konrad Zuse, J. Presper Eckert, John Mauchly, Tom Kilburn, Tommy Flowers, Howard Aiken und Maurice Wilkes. Wer sich auf Konzepte statt auf tatsächlich funktionierende Maschinen konzentriert, kann dasselbe mit Charles Babbage und John von Neumann anstellen – und wird das auch tun. Einer meiner Kollegen meinte einmal scherzhaft, wir sollten den ersten Computer ausfindig machen, der noch nie als der erste Computer bezeichnet worden ist.

Die Geschichte hinter all diesen ‹Erstzuschreibungen› lautet ungefähr wie folgt: Von den späten 1930er Jahren bis Mitte der 1940er wurde eine ganze Reihe automatischer Rechenmaschinen gebaut. Ihre Erfinder wussten oft nichts voneinander. Manche nutzten elektromechanische Relais für ihre Logikschaltungen, während andere auf der Basis von Elektronenröhren bauten. Daneben gab es Maschinen, die Abfolgen von Anweisungen von Lochstreifen ablasen. Es ist u. a. einer Reihe von juristischen Auseinandersetzungen um ein Patent der[1] ENIAC zu verdanken, dass diese Maschinen die frühe Diskussion der Geschichte des Computers dominierten und dass ihre Erfindung gut dokumentiert ist.

1 In der deutschsprachigen Forschungsliteratur werden Anlagen wie EDVAC und ENIAC im Maskulinum oder im Femininum geschrieben. Die unterschiedliche Verwendung ist auf die Frage rückführbar, ob man ENIAC, EDVAC und andere als *Anlagen* oder als *Rechner* bzw. *Computer* auffasst. Die Übersetzung schließt im Folgenden historisierend an die Epoche der «Rechenanlagen» an. Anm. d. Ü.

Die ‹modernen› oder ‹speicherprogrammierten› Rechner, aus denen sich die nachfolgenden Computer entwickelten, wurden durch zwei Durchbrüche definiert, die miteinander zusammenhängen. Auf technischer Ebene entschied sich der Erfolg und das Scheitern der Computer-Projekte in den späten 1940er Jahren an der Fähigkeit, große, schnelle Speicher zuverlässig zum Laufen zu bekommen. Die erste hierfür vorgeschlagene Technologie war die Quecksilber-Verzögerungsleitung und kam von dem Leiter der Entwicklung der ENIAC an der University of Pennsylvania, J. Presper Eckert. Freddy Williams, der im Computer-Projekt der Universität Manchester arbeitete, war der erste, der die Speicherung auf einer Kathodenstrahlröhre zustande brachte. Diese Möglichkeiten wurden zu den beiden dominanten Hochgeschwindigkeits-Speichertechnologien bis Mitte der 1950er Jahre.

Der Durchbruch auf konzeptueller Ebene war die Erfindung dessen, was wir heute Computerarchitektur nennen, und mit ihr die Nutzung der Flexibilität, die aus den neuen Speichertechnologien hervorging. Historiker sind sich darüber einig, dass alle Computer, die in den späten 1940er Jahren auf der Welt gebaut wurden, von einem einzigen Konzept inspiriert wurden: einem unveröffentlichten Typoskript mit dem kryptischen Titel «First Draft of a Report on the EDVAC». Dieses unvollständige Dokument fasste Diskussionen der Gruppe zusammen, die an einer Nachfolgerin der ENIAC arbeitete. Auf seiner Titelseite wird ausschließlich John von Neumann als Verfasser genannt; es ist allerdings umstritten, inwieweit dort wirklich seine eigenen Ideen festgehalten sind oder ob er selbst vielmehr lediglich den Fortschritt der Gruppendiskussionen dokumentierte. Turing hat die Konstruktion für seine Automatic Computer Engine (ACE) unter dem Lektüreeindruck dieses Dokuments verfertigt, auch wenn sein Ansatz in verschiedenen interessanten Hinsichten von dem von Neumanns abweicht.

In den 1940er Jahren waren Turings Arbeiten nicht ganz unbekannt. Es gibt zum Beispiel stichhaltige Hinweise, dass von Neumann von einem heute berühmten Dokument Turings wusste und dessen Interessen an den dahinterliegenden mathematischen Fragen teilte. So könnte man darüber spekulieren, ob das Bündel an Ideen in von Neumanns «First Draft» nur eine Wiederholung – oder im besten Fall eine Ausarbeitung – von Turings früheren Arbeiten über Berechenbarkeit ist. Urteilen Sie darüber selbst, indem Sie Turings «On Computable Numbers …» von 1936 und den «First Draft of a Report on the EDVAC» nebeneinander legen. Sie sind zwar ohne weiteres über Google zu finden, allerdings sollten Sie sich vielleicht erst ein stärkendes Getränk genehmigen, denn keines der beiden ist sonderlich leicht lesbar. Ersteres ist ein Aufsatz über mathematische Logik. Er beschreibt ein Gedankenexperiment, so wie der Physiker Schrödinger mit seiner berühmten Katze aus dem Jahre 1935, die sich gefangen in einer Kiste zwischen Leben und Tod bewegt – abhängig vom Verhalten eines einzelnen Atoms. «Schrödingers Katze» ist kein Aufsatz über Katzen oder über die richtige Behandlung von Katzen. Und genauso

wenig ging es Turing darum, die Konstruktion einer neuen Art von Rechenmaschine vorzuschlagen.

Wie der Titel des Aufsatzes nahelegt, hatte Turing seine genialen imaginären Maschinen entworfen, um die Frage nach den fundamentalen Grenzen mathematischer Beweisbarkeit aufzuwerfen. Sie waren zum Zwecke der Einfachheit gestaltet und hatten wenig mit den Ansätzen jener gemein, denen es um den Bau tatsächlicher Maschinen ging. Von Neumanns «Draft» hingegen enthält keine explizite Aussage über mathematische Logik. Der Bericht beschreibt die Architektur eines geplanten Rechners, kommt dann zu Technologien, mit denen dies realisiert werden könnte, und dient insgesamt der Aufgabe, die Gruppe anzuleiten, die bereits den Auftrag erhalten hatte, die EDVAC zu entwickeln.

Von Neumann geht auf die Details der Hardware nicht ein, einerseits um sich stattdessen auf das zu konzentrieren, was wir heute ‹Architektur› nennen würden, andererseits weil die an der Moore School laufenden Computer-Projekte im Jahr 1945 immer noch der Geheimhaltung unterlagen. Seine Briefe aus dieser Zeit sind voll von Beschreibungen technischer Details, wie Skizzen spezieller Elektronenröhrenmodelle und deren Leistungsdaten. Die Formulierung «stored program concept», das «Prinzip der Speicherprogrammierung» ist manchmal verwendet worden, um den Inhalt des «First Draft» schlagwortartig zu verdichten, aber wenn man damit sagen will, dass dieser Bericht nur eine große Idee zu bieten hatte, wird man seiner tatsächlichen Bedeutung nicht gerecht. Bei näherer Betrachtung findet sich dort eine Fülle ineinander verschlungener Ideen und Details. Für meine derzeitigen Arbeiten mit Mark Priestley und Crispin Rope habe ich diese in drei Felder unterteilt.

Das erste, das *EDVAC-Hardware-Paradigma*, beschreibt einen vollelektronischen binären Computer mit einem Speicher, der größer ist als alles, was bis dahin gebaut worden war.

Das zweite, das *Leitbild der von-Neumann-Architektur*, formuliert die Grundstruktur des modernen Computers: Spezialregister, durch die alle Operationen ausgeführt und von denen ausgehend Daten mit dem Hauptspeicher ausgetauscht werden, Trennung arithmetischer Funktionen von Steuerfunktionen und Speichereinheiten, immer nur eine Aktion wird zu derselben Zeit ausgeführt und so weiter.

Das dritte, das *moderne Code-Paradigma* betrifft die Art und Möglichkeiten der Eingaben. Sie wurden beispielsweise durch ein schmales Vokabular von Befehlscodes ausgedrückt, gefolgt von Argument- oder Adressfeldern und in den gleichen nummerierten Speicherzellen festgehalten wie die Daten. Während die Maschine standardmäßig einen bestimmten Ablauf verfolgte, konnte sie aus diesem Ablauf heraus springen, und das Ziel dieses Sprungbefehls konnte, während das Programm lief, in Abhängigkeit vom Stand der Berechnung modifiziert werden.

II. Die Geschichte des Prinzips der Speicherprogrammierung

Trotz der Rolle, die ihm in späteren historischen Arbeiten zugeschrieben wurde, ist der «First Draft of a Report on the EDVAC» nicht etwa eine Art Normierung, die das Prinzip der Speicherprogrammierung im Detail darlegt. Vor allem aber erscheint an keiner Stelle das Wort «program». Von Neumann bevorzugte konsequent «code» vor «program» und schrieb stets «memory» statt «storage».

Unsere derzeitige Neigung zum Begriff *Speicherprogrammierung* als Beschreibung für Computer, die nach dem für die EDVAC vorgeschlagenen Grundprinzip gebaut wurden, bedarf daher einer historischen Erklärung. Wörtlich genommen besagt der Begriff ziemlich wenig. Jedes von einem Computer ausgeführte Programm muss in irgendeinem Medium gespeichert werden. Der «First Draft» selbst beobachtet, dass

> […] die Anweisungen in einer Form gegeben werden müssen, die das Gerät wahrnehmen kann. In Lochkartensysteme oder auf das Band eines Fernschreibers gestanzt, magnetisch aufgezeichnet auf Stahlband oder Draht, fotografisch aufgezeichnet auf Zelluloidfilm, verdrahtet in eine oder mehrere feste oder austauschbare Schalttafeln – wobei diese Liste keineswegs notwendigerweise komplett ist.[2]

Der «First Draft» argumentiert dafür, Code und Daten gemeinsam zu verorten, wenn auch nur zaghaft:

> Während es so schien, dass verschiedene Teile dieses Speichers [*memory*] Funktionen ausführen müssen, die sich in ihrem Zweck unterscheiden, so ist es doch verlockend, den gesamten Speicher als ein Organ zu behandeln und seine Teile so austauschbar wie möglich zu halten.[3]

Von Neumann war überzeugt, dass die Datenanforderungen der Probleme, auf die sich sein Interesse richtete, einen Speicher in bis dato ungekannter Größe verlangten, und dass der Programmcode im Vergleich dazu ziemlich klein sein würde. Es würde die EDVAC vereinfachen, wenn man beide mit demselben Set von Mechanismen bedienen könnte.

Der «First Draft» zirkulierte schnell zwischen jenen, die am Bau von Computern interessiert waren, und etablierte sich auf diesem Wege ohne Verzögerung als Vorlage für die folgenden Computer-Projekte. Die frühen Diskussionen des EDVAC-Ansatzes befassten sich zwar mit einer ganzen Reihe innovativer Funktionen; dass aber die spezielle Methode der Programmspeicherung die wichtigste darunter war, war keineswegs ausgemacht. Mehrere Einführungsbücher zu dem Thema befassten sich in erster Linie mit der Programmiermethode und dokumentierten den Befehlssatz des Computers, den von Neumann am Institute for Advanced Studies geplant hatte.

Für viele der Computerbauer der 1940er Jahre, darunter die ENIAC-Erfinder J. Presper Eckert und John Mauchly, deren Beitrag zur neuen Konstruktionsweise

2 John von Neumann: First Draft of a Report on the EDVAC, in: IEEE *Annals of the History of Computing*, Jg. 15, Nr. 4, 1993, 27–75, hier Section 1.2.

3 Ebd., Section 2.5.

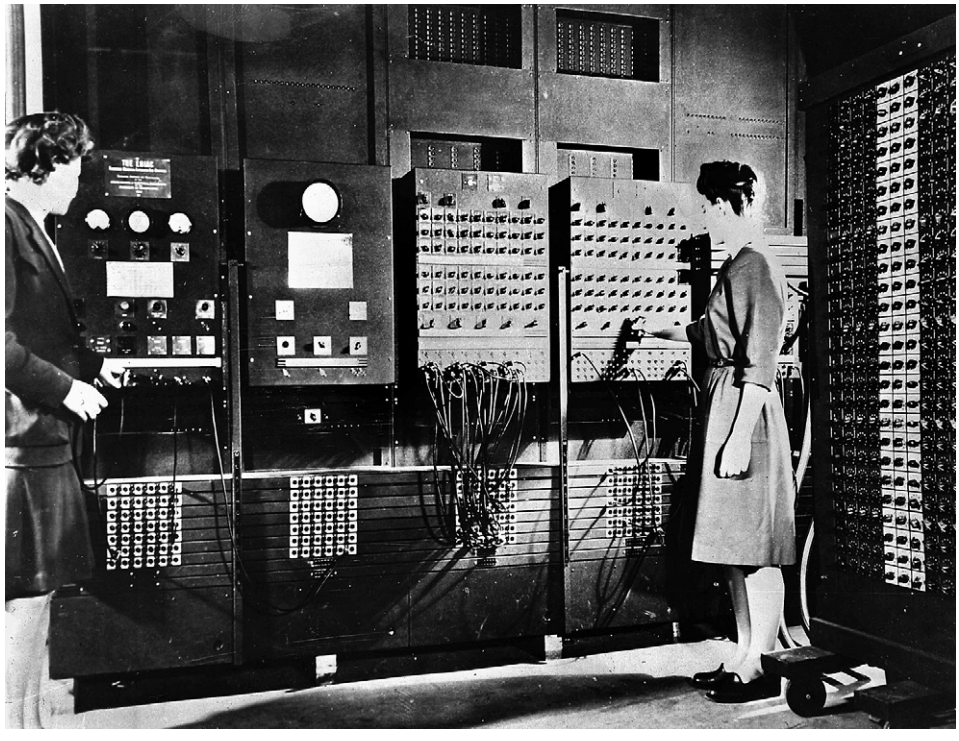

Abb. 1 Operateurinnen an einer ENIAC-Rechenmaschine, USA 1946

grundlegend war, lag der Vorteil des sogenannten EDVAC-Type-Ansatzes in der Demonstration der Möglichkeit, wie man mit relativ wenigen der teuren und unzuverlässigen Elektronenröhren einen leistungsfähigen und flexiblen Computer baut. Die Schwierigkeit der Ingenieursleistung bestand in diesem Fall darin, einen großen Hochgeschwindigkeitsspeicher anzufertigen. Die leichte Programmierbarkeit und der schnelle Wechsel von einem Problem zum nächsten wurden dabei als die Vorteile dieses Ansatzes betrachtet; und dieser Vorteil stand im Zusammenhang damit, dass Code und Daten in demselben elektronischen Speicher vorgehalten wurden.

Den Ausdruck «stored program» konnten wir in keinem der Computerkonferenzberichte oder Lehrbücher finden, die in den 1940er Jahren publiziert worden sind. Seine früheste bekannte Verwendung tritt im Jahr 1949 auf und stammt von einem kleinen Team vom IBM-Standort in Poughkeepsie, das einen Testaufbau (*test assembly)* baute, der IBMs ersten Rechner nach EDVAC-Bauart darstellt. Dieses experimentelle System wurde um die erste elektronische Rechenmaschine des Unternehmens herumgebaut, den elektronischen Rechenlocher IBM 604, der jetzt als Rechenwerk dieses zusammen gebastelten Computers diente. Dazu baute die Mannschaft ein neues Steuerwerk, einen Kathodenstrahlröhrenspeicher und einen Trommelspeicher.

Ein interner Vorschlag von Nathaniel Rochester aus dem Jahre 1949[4] bemerkt, dass der Stecktafel-Ansatz mit großen Programmen nicht machbar sei,

4 Nathaniel Rochester: *A Calculator Using Electrostatic Storage and a Stored Program* [1949], in: IBM's Early Computers Sources Collection, IBM Archives Zitat 17. May 1949.

was aber dadurch gelöst werden könne, dass man «das Rechenprogramm in die Maschine über ein Deck von Karten einliest und es zusammen mit den numerischen Daten in der Speichersektion der Rechenmaschine behält». Rochesters Dokument trug den Titel «A Calculator Using Electrostatic Storage and a Stored Program».[5]

Bei IBM entwickelte sich die Bedeutung von «Speicherprogrammierung» bald von der wörtlichen Beschreibung eines bestimmten Programmiermechanismus zu einer allgemeinen Beschreibung von Maschinen nach EDVAC-Bauart. In den 1950er Jahren taucht der Begriff manchmal in Tagungsberichten auf, besonders solchen von IBM-Mitarbeitern, aber da alle leistungsfähigen Computer bald diesem Vorbild folgten, sprach man meist einfach von ‹Large-Scale Digital Computer›.

III. Die Geschichtsschreibung der «Speicherprogrammierung»

Seit Ende der 1970er Jahre ist die Computergeschichte als akademische Teildisziplin entstanden. Zu Beginn konzentrierte man sich auf die Maschinen der 1940er Jahre. Nach dem Vorbild des Historikerpioniers Herman Goldstine entlehnte man den Begriff des «Speicherprogrammierten Rechners» dem technischen Diskurs, und nachdem er zuerst recht obskur erschienen war, wurde er zum zentralen Begriff hitziger Debatten um die Frage, was warum als erster echter Computer anzusehen sei.

Grob vereinfacht wurde man diese kaum lösbare und wenig zielführende Streitfrage dadurch los, dass man sich auf ein paar Wortketten einigte, die jeweils klarstellten, was genau welche der frühen Maschinen als erste zu Stande gebracht hatte. Die ENIAC zum Beispiel wurde so zum ersten «Large-Scale General-Purpose Digital Electronic Computer to be Fully Operational». Die EDSAC aus Cambridge und das «Baby» von der Universität Manchester wurden als die «ersten funktionsfähigen, nach EDVAC-Bauart strukturierten speicherprogrammierten Rechner» anerkannt, aber es gab wenig Notwendigkeit oder Interesse daran, den ‹Begriff› genauer zu definieren, um seine notwendigen und hinreichenden Merkmale zu identifizieren.

In der jüngsten Vergangenheit haben Computerhistoriker ihre Aufmerksamkeit weitgehend von den 1940er Jahren abgewandt, nachdem man einen Konsens über die Ehrentitel erreicht hatte, die jeder dieser frühen Maschinen zuerkannt werden konnten. Über die Ereignisse der 1940er ist ein weithin akzeptiertes Narrativ entstanden. Wie Swade bemerkt, wird das Prinzip der Speicherprogrammierung in akademischen und populären Geschichtsschreibungen weiterhin als entscheidender Schritt verzeichnet, gleichzeitig aber sehr wenig analysiert.[6]

Mit dem Aufkommen der theoretischen Informatik und einer neuen Begeisterung für die Arbeit an abstrakten Rechnermodellen fasste man den digitalen Computer seit den 1950er Jahren mehr und mehr als eine Verkörperung der

5 An dieser Stelle danke ich Peggy Kidwell dafür, dass sie mich auf dieses Dokument aufmerksam gemacht hat.

6 Doron Swade: Inventing the User. EDSAC in Context, in: *The Computer Journal*, Jg. 54, Nr. 1, 2011, hier 146.

universellen Turingmaschine auf. Der entscheidende Vorteil des Computers war es, Anweisungen als Daten zu behandeln und sie durch Programmierung modifizieren zu können. Die von Swade dennoch konstatierte Verwirrung scheint auf Meinungsverschiedenheiten zwischen einer Partei, die vom pragmatischen Vorgehen der 1940er Jahre beeinflusst war und der anderen, die sich vornehmlich mit den später gestellten, theoretischen Fragen befasste, zurückzugehen.[7] Mit dem Wiederaufleben des Prinzips der Speicherprogrammierung als historiografischem Begriff identifizierte man diese mehr und mehr mit grundlegenden Errungenschaften der Informatik.

In den vergangen Jahrzehnten wurde die Möglichkeit, Daten und Programme gleichermaßen in denselben Speichereinheiten zu bearbeiten, als entscheidendes Charakteristikum speicherprogrammierter Rechner und damit auch moderner Computer betrachtet. Im Gegenzug wurden die Begriffe «speicherprogrammierter Rechner» und «Allzweck-Rechner» manchmal mit dem formaleren Begriff des «universellen» oder «potenziell turingmächtigen Rechners» vermischt – was beide Male bedeutet, dass er turingmächtig wäre, würde er über einen unendlich großen Speicher verfügen.

Um nur drei der vielen Beispiele für diese konventionelle Weisheit zu zitieren: Der Wikipedia-Eintrag zu «Stored-program computer» definiert ihn derzeit als einen, «der Programmanweisungen in einem elektronischen Speicher speichert. Diese Definition wird oft um die Anforderung erweitert, dass die Bearbeitung von Programmen und Daten nach den gleichen Prinzipien möglich sein muss […]. Die Idee des speicherprogrammierten Rechners lässt sich bis zur universellen Turingmaschine aus dem Jahr 1936 zurückverfolgen.» In seinem relativ neuen Buch *Computing: A Concise History*[8] definiert Paul Ceruzzi speicherprogrammierte Rechner als solche, die «ihre Anweisungen – die Programme – und die Daten speichern, mit denen diese Anweisungen in demselben physischen Speicher arbeiten», und deutet an, dass diese Eigenschaft «Turings Ideen in die Gestaltung praktischer Maschinen einbrachte.» Sogar Swade selbst kam zu der eher konventionellen Schlussfolgerung, dass «das intern gespeicherte Programm […] die praktische Realisierung der Turing-Universalität» sei, und es verleihe «den Funktionen jene Plastizität, die zum größten Teil für die bemerkenswerte Verbreitung von Computern und Computer-artigen Artefakten verantwortlich» sei.[9]

Es ist für Computerhistoriker zu einem amüsanten Gesellschaftsspiel geworden, über den Einfluss zu streiten, den Turing auf von Neumann hatte oder nicht hatte. Abgesehen davon findet man in den Diskussionen der Leute, die in den 1940er Jahren Rechner gebaut haben, sehr wenige Bezüge zu Turings theoretischer Arbeit.[10] Atsushi Akera hat vorgeschlagen, die rückwirkende Umarmung Turings im Zusammenhang mit dem Hang der Informatik zu abstrakten Rechnermodellen zu betrachten, der sie in den späten 1950er und 1960er Jahren erst zu einer eigenständigen Disziplin gemacht hat.[11] In späteren Diskussionen wurden die Vorteile speicherprogrammierter Maschinen oft anhand

7 Ebd.

8 Paul E. Ceruzzi: *Computing. A Concise History*, Cambridge, Mass. 2012.

9 Swade: Inventing the User, 146. s.o.

10 Mark Priestley: *A Science of Operations. Machines, Logic, and the Invention of Programming*, New York 2011; Simon Lavington, Chris Burton, Martin Campbell-Kelly (Hg.): *Alan Turing and His Contemporaries. Building the World's First Computers*, Swindon 2012.

11 Atsushi Akera: *Calculating a Natural World. Scientists, Engineers, and Computers During the Rise of U.S. Cold War Research*, Cambridge, Mass. 2006.

theoretischer Fragen jüngerer Provenienz begründet, statt mit den praktischen Problemen, die für die damaligen Konstrukteure vordringlich waren.

In diesem Sinne kann uns die Suche nach den logischen und historischen Grundlagen der Computer in zwei entgegengesetzte Richtungen ziehen. Es war die besondere Klarheit und Abstraktion von den unordentlichen Details der Hardware, die der Turingmaschine ihren legendären Ort in der Informatik verschafft hatte. Die Ideen der Turingvollständigkeit bzw. Turingmächtigkeit und der universellen Turingmaschine dienten dazu, die theoretische Informatik von der materiellen Welt der Plattformen und Architekturen zu entkoppeln.

Wenn zum Beispiel einmal gezeigt worden war, dass ein in Conways Spiel des Lebens gebauter virtueller Computer äquivalent zu einer universellen Turingmaschine ist, zeigte uns das bereits, dass dieser Computer, mit genügend Zeit und entsprechend großem Spielfeld, dieselben Algorithmen ausführen kann wie jede aus herkömmlichen Komponenten gebaute Maschine. Der intellektuelle Nutzen dieses Ansatzes liegt auf der Hand, ebenso wie sein strategischer Vorteil für die frühe Informatik-Community in einer Zeit, in der sie darum kämpfte, sich von der Mathematik und Elektrotechnik zu lösen, um den Status einer Hilfswissenschaft hinter sich zu lassen.

Der unlängst verstorbene Michael Mahoney bemühte sich lange Jahre darum, die Geschichte der theoretischen Informatik nach außen hin abzugrenzen.[12] Sein großes Thema war das Bedürfnis von Scientific Communities, ihre eigenen historischen Narrative zu konstruieren. Mahoney betrachtete die theoretische Informatik als eine Ansammlung mathematischer Tools, die eigentlich in ziemlich unterschiedlichen Kontexten entwickelt worden waren, von Gruppentheorie und Lambda-Kalkül bis zur Chomsky-Hierarchie formaler Grammatiken. In einem etwas weiteren Horizont betrachtet, hatten sowohl Arbeiten zur mathematischen Logik als auch die Konstruktion von Rechenmaschinen eine lange, aber getrennte Geschichte. Doch innerhalb der Disziplin und aus heutiger Sicht schienen die Verbindungen zwischen den beiden einfach zu offensichtlich und die Geschichte wird oft so geschrieben, als ob die Resultate der entsprechenden Arbeiten jahrhundertelang auf die Entwicklung des Computers hinauslaufen und als ob die Computerpioniere der 1940er Jahre in erster Linie von Turing beeinflusst gewesen seien.

Wie Mahony schreibt, birgt das Interesse der Praktiker, «ihre eigene Geschichte zu finden […], eine echte Gefahr», denn obwohl wissenschaftliche Historiker und Praktiker «eine gemeinsame Geschichte suchen», geschehe dies «nicht mit demselben Zweck und nicht von demselben Standpunkt aus».[13] Abstraktion ist die Seele der Informatik, aber als Historiker verlieren wir das Lebenswichtigste, wenn wir von der historischen Schlampigkeit früher Computer-Projekte abstrahieren, ihrem Fokus auf Ingenieursfragen, ihren spezifischen Zielen und Wurzeln im Denken der 1940er Jahre.

Die Abstraktion von tatsächlichen Computern und tatsächlicher Rechenpraxis, die mit der Konzentration auf Turingmächtigkeit einhergeht, ist gut

12 Michael Sean Mahoney: *Histories of Computing*, hg. u. m. Einl. v. Thomas Haigh, Cambridge, Mass., London 2011.

13 Michael S. Mahoney: Finding a History for Software, IEEE Annals of the History of Computing, Jan./Mar. 2004, 8–19, hier 9.

für die Theorie, aber schlecht für die Historiografie. Der Versuch von Raúl Rojas zum Beispiel, zu beweisen, dass Zuses Z3 aus dem Jahre 1943 ein Universalrechner gewesen sei oder als solcher hätte umgesetzt werden können,[14] ist zwar ein beeindruckender Party-Trick, aber hat nichts damit zu tun, wie die Maschine konstruiert und tatsächlich genutzt wurde oder auch nur sonst mit irgendwas, das in den 1940er Jahren sinnvoll gewesen wäre. Das beschriebene Programmierverfahren hätte irrwitzig lange Papierstreifen gebraucht und einen massiven Verlust an Rechenleistung bedeutet. Es wäre wohl schneller geblieben, alles von Hand auszurechnen. Die eigentliche Lehre scheint mir zu sein, dass die Z3 mit nur einer kleinen Konstruktionsänderung turingmächtig hätte sein können, sie war es aber nicht, weil das Prinzip und sein Nutzen nicht von vielen verstanden wurden. In der Tat behauptete Zuse später, im Laufe der Konstruktion in Betracht gezogen und wieder verworfen zu haben, Programmanweisungen als Daten zu behandeln. Die Vergangenheit bleibt tatsächlich ein befremdliches Land. Immerhin konnte Rojas mit seiner Verbeugung vor der Welt der Theorie den Status von Zuses Maschine und den deutschen Stolz weiter heben.

In diesem Zusammenhang sollte man auch erwähnen, dass von Neumanns «First Draft» aus dem Jahre 1945 ausdrücklich uneingeschränkte Code-Änderungen verbietet, eine bemerkenswert grundsätzliche Abweichung von dem, was heute als Turingmaschinen-Modell des Universalrechners verstanden wird. Die dort beschriebene EDVAC brauchte für viele gängige Operationen Code-Änderungen, auch für einen Schleifenabbruch und andere Steuerkonstrukte, und für eine Änderung der Adresse, von der Daten abgerufen werden (zum Beispiel, um jedes Mal, wenn ein Codeabschnitt durchlaufen wurde, einen Wert von einer anderen Zelle innerhalb eines Arrays zu erhalten). Darin spiegelt sich eine weiterreichende Konstruktionsphilosophie, die auf radikale Vereinfachung der Computerarchitektur setzt, indem man die bei früheren Konstruktionen üblichen Spezialmechanismen durch eine kleine Anzahl von Allzweckmechanismen ersetzt.

Von Neumanns Befehlssatz für die EDVAC jedenfalls verhinderte es explizit, komplett überschrieben zu werden.[15] Nur die Adressfelder konnten geändert werden. Eines der 32 Bits in jedem Datenwort kennzeichnete, ob es Daten oder Programm enthielt. Mit einer Anwendung einer Transferoperation auf ein Befehlswort würde man lediglich das Adressfeld überschreiben.

Zeit für ein Zwischenfazit. Die Diskussion der Speicherprogrammierung hat mittlerweile den Zweck überlebt, für den das Prinzip selbst erfunden wurde und trägt jetzt nur noch zur historischen Verwirrung bei. Es ist an der Zeit, es als analytische Kategorie zu ersetzen, und zwar mit einigen spezifischeren Alternativen, mit denen klare und präzise Definitionen möglich sind. Im Folgenden werde ich drei Vorschläge zur Ersetzung machen, den ersten etwas detaillierter, die beiden anderen in Form einer Skizze.[16]

14 Raúl Rojas: How to Make Zuse's Z3 a Universal Computer, in: IEEE *Annals of the History of Computing*, Jg. 20, Nr. 3, 1998, 51–54.

15 Priestley: *A Science of Operations*; Michael D. Godfrey, David F. Hendry: The Computer as von Neumann Planned It, in: IEEE *Annals of the History of Computing*, Jg. 15, Nr. 1, 1993, 11–21.

16 Vgl. die ausführlicheren Darstellungen der Forschergruppe *ENIAC in Action* (Thomas Haigh, Mark Priestley, Crispin Rope), aktuelle Literaturangaben unter *eniacinaction.com*.

IV. Das moderne Code-Paradigma

Der erste ist das *moderne Code-Paradigma*. Dieser neue Begriff beschreibt die programmbezogenen Elemente der 1945er «First-Draft»-Konstruktion, die zur Grundausstattung der Computerkonstruktionen der 1950er Jahre wurden. Manches, was in diesem Bericht dargelegt wurde, hat man ignoriert oder geändert (so etwa das Fehlen eines eigenen bedingten Verzweigungsbefehls), während andere übliche Codefunktionen der 1950er Jahre-Computer (wie beispielsweise Index-Register) aus anderen Quellen stammten.

Im «First Draft» nach neuen Funktionen zu suchen, die mit Code zu tun haben und ein Jahrzehnt später selbstverständlich geworden sein sollten, wirft ein Schlaglicht auf den umfassenderen Prozess, durch den ein überbordendes, eigenwilliges und brillantes Dokument zu einem dominanten Paradigma für die Computerbauer wurde. Während der theoretische Informatiker nach den minimal erforderlichen Funktionen für Universalität schaut und diese bei Turing findet, geht es mir hier darum, die maximale Menge von Funktionen zu finden, die bereits im 1945er «First Draft» beschrieben wurden und es dann historisch in die Standardkonstruktionen der 1950er Jahre geschafft haben.

1. *Das Programm wird komplett automatisch ausgeführt.* Um den «First Draft» zu zitieren: «Sind die Befehle in das Gerät eingegeben, muss es in der Lage sein, sie komplett und ohne weitere intelligente menschliche Intervention auszuführen.»[17] Dies war notwendig für elektronische Maschinen, während händische Intervention an Verzweigungspunkten bei langsameren Geräten wie dem Harvard Mark I durchaus machbar waren.
2. *Das Programm wird als einzelne Abfolge von Anweisungen geschrieben, oder «Befehlen», wie es im* «First Draft» *heißt, die zusammen mit den Daten an nummerierten Speicherstellen gespeichert werden.* Diese Befehle steuern alle Details des Betriebs. Um Daten und Code zu lesen, werden dieselben Mechanismen genutzt. Wie bereits erwähnt, beschreibt der «First Draft» im Detail die explizite Trennung von Speicherstellen, die Code beinhalten, von jenen, wo Daten gespeichert sind. Es gibt dort auch bereits Hinweise auf die Idee von Programm als lesbarem Klartext: «Es ist in der Regel bequemer, wenn die kleineren Zyklen, die die aufeinanderfolgenden Schritte in einer Sequenz logischer Befehle ausdrücken, automatisch aufeinander folgen.»[18]
3. *Jeder Befehl innerhalb des Programms ist eine Einzeloperation, die als Teil der logischen Einheit* atomarer Operationen *fungiert, von denen dem Programmierer eine bestimmte Menge zur Verfügung steht.* Dabei begann man den Befehl normalerweise mit ein paar wenigen Opcodes, gefolgt von Argumentfeldern, die einen Speicherort oder andere Parameter auszeichnen. Insgesamt brauchten Befehle neun bis 22 Bits. Tatsächliche Maschinen folgten in der Regel diesem Muster. Die wichtigste Ausnahme bildet

17 Von Neumann: First Draft, Section 1.2.

18 Ebd., Section 14.4.

Turings ACE-Konstruktion und deren Unterformen, die nah an der Hardware blieben, indem sie alle Befehle als Datentransfers zwischen Quellen und Zielen codierten.

4. *Die Befehle des Programms werden normalerweise in einer vorgegebenen Reihenfolge ausgeführt.* Laut «First Draft» sollte die Maschine «angewiesen werden, nach jedem Befehl den nächsten zu finden, den sie auszuführen hat.»[19] In der EDVAC wurde dies einfach implizit durch die Reihenfolge bewerkstelligt, in der sie gespeichert waren, wie in «normaler Routine» sollte sie «den Befehlen in der zeitlichen Reihenfolge gehorchen, in der sie natürlicherweise auftauchen».[20]
5. *Allerdings kann ein Programm den Computer anweisen, von dieser normalen Reihenfolge abzuweichen und zu einer anderen Stelle im Programm zu springen.* «Es müssen jedoch Befehle zur Verfügung stehen, die in den erwähnten Ausnahmefällen den CC anweisen, seine Verbindung zu jedem anderen beliebigen Punkt im Speicher zu transferieren [d.h. die nächste Anweisung von diesem zu holen].»[21] Dies ermöglichte Funktionen wie Sprunganweisungen und Unterprogramme, die man aufrufen und zurück verzweigen kann.
6. *Die Adresse, auf die sich ein Befehl bezieht, kann sich im Laufe der Programmausführung ändern.* Das gilt für die Quelle oder das Ziel der Daten für Berechnungen oder das Ziel eines Sprungs. Diese Adressänderungsfähigkeit kommt im «First Draft» in ziemlich kryptischer Form vor, dessen letzter Satz bemerkt, dass wenn eine Zahl zu einer Speicherstelle transferiert wird, die einen Befehl enthält, nur die letzten 13 Stellen, die die Adresse μ0 repräsentieren, überschrieben werden sollen. Tatsächliche Computer erzielten funktional äquivalente Fähigkeiten durch eine Kombination aus uneingeschränkten Codeänderungen, indirekten Adressierungsmechanismen und bedingten Verzweigungsbefehlen.

Eine Konsequenz der hier beschriebenen Konstitution des Codes lag darin, dass die logische Komplexität des Programms nur vom Speicherplatz begrenzt war, der zur Verfügung stand, um Anweisungen und Arbeitsdaten zu speichern. Dies stand im Gegensatz zur Abhängigkeit von Maschinen wie der ursprünglichen ENIAC oder der SSEC von einer Vielzahl von Ressourcen wie Programmzeilen, Stecktafelkapazität oder Lochstreifenlesern als potenziellen Begrenzungen für logische Programmkomplexität.

V. Die Von-Neumann-Architektur und das EDVAC-Hardware-Paradigma

Das *moderne Code-Paradigma* soll kein neuer Name für das Prinzip der Speicherprogrammierung sein oder als Idee dienen, um die mit diesem Begriff assoziierten Bedeutungen in vollem Umfang mit zu umfassen. In der Tat liegt die Anziehungskraft des Begriffs der Speicherprogrammierung in seiner Spezifität. Es gab aber ganz eindeutig auch noch andere Aspekte des «First Draft»

19 Ebd., Section 14.3.
20 Ebd., Section 14.3.
21 Ebd., Section 14.3.

und seiner Folgepublikationen aus dem Kreise der Von-Neumann-Gruppe in Princeton, die einen großen Einfluss auf spätere Computerbauer hatten.

Um einen bestehenden Begriff zu nutzen, könnte man eine dieser Facetten das *Paradigma der Von-Neumann-Architektur* nennen. Es beinhaltet die Grundstruktur von ‹Organen›, die im «First Draft» charakterisiert werden, inklusive der Trennung des Speichers auf der einen Seite und der Steuerung und Arithmetik auf der anderen. Damit verbunden ist die Serialisierung der Rechenvorgänge, die bewirkt, dass zu einem Zeitpunkt immer nur eine Operation abläuft und das Routing aller Speichertransfers durch die zentrale Recheneinheit stattfindet. Auch das System der Sonderzweckregister, die als Quelle und Ziel für arithmetische und logische Anweisungen dienen und einen Befehlszähler und Befehlsregister für Steuerungszwecke nutzen, ist Teil dieses Paradigmas. In der Fachliteratur wurde die Von-Neumann-Architektur in der Regel deutlich klarer definiert als das Prinzip der Speicherprogrammierung. Man könnte, so wie manche dies auch getan haben, diskutieren, inwieweit es gerechtfertigt ist, nur von Neumanns Namen im Zusammenhang mit diesen Prinzipien und Begriffen zu nennen. Ein alternativer Name für die Von-Neumann-Architektur wäre das *Paradigma der EDVAC-Architektur*.

Die dritte wichtige Facette könnte das *EDVAC-Hardware-Paradigma* genannt werden. Der EDVAC-Ansatz fand vor allem auch deshalb Anklang bei den damaligen Computerbauern, weil er eine Möglichkeit bot, leistungsfähige, flexible Maschinen zu bauen, ohne dabei allzu viele Komponenten zu benötigen. Zu den einflussreichen Hardware-Ideen gehört etwa der Gebrauch von Speicherröhren oder auch von Verzögerungsleitungen zur Laufzeitspeicherung, die Logik nur aus elektronischen Komponenten bauen, alle möglichen Quantitäten in Binärcode abbilden und doppelte oder spezielle Hardware-Mechanismen möglichst gering halten (von Neumann war der Auffassung, dass ein Multiplizierer gerechtfertigt sei, aber dass es wenig Nutzen brächte, einen Volladdierer zu duplizieren oder Hardware für spezialisiertere Funktionen zu nutzen). Mit Ausnahme der erwähnten Speichertechnologien waren diese Hardwarefunktionen nicht ungewöhnlich, aber in der Gesamtheit zeigten sie ein mutiges Bekenntnis zu neuen Technologien zu einer Zeit, als Forschergruppen in Harvard, bei Bell Labs und IBM immer noch Pläne für neue High-End-Maschinen auf der Basis von Relais-Speichern und Lochstreifensteuerung aufzeichneten. Wir sind daher überzeugt, dass die im «First Draft» dargelegten Hardware-Entscheidungen ein Paradigma darstellen, und zwar ein materiell greifbares und leistungsfähiges Leitbild in Thomas Kuhns Sinne.[22]

VI. Unterschiedliche Wege

Diese drei Leitbilder haben miteinander verflochtene Frühgeschichten, waren aber immer zumindest teilweise separierbar, und entwickelten sich schließlich ganz auseinander. Viele Maschinen der 1940er Jahre implementieren manche

22 Thomas S. Kuhn: Second Thoughts on Paradigms, in: ders.: *The Essential Tension: Selected Studies in Scientific Tradition and Change*, Chicago 1979, 293–319.

Aspekte der EDVAC-Paradigmen, aber andere nicht. Alan Booths ARC folgte sowohl dem Paradigma des modernen Codes als auch der Von-Neumann-Architektur, aber implementierte sie mit Relais-Hardware. Martin Campbell-Kelly hat die Beobachtung gemacht, dass Booths angeblicher erster Betriebstag – der 12. Mai 1948 – diesen zum «ersten betriebsfähigen speicherprogrammierten Rechner nach EDVAC-Bauart» machen würde, und fügt in Klammern hinzu: «auch wenn er natürlich nicht elektronisch war».[23] Alan Turings Konstruktion für die ACE übernahm von Neumanns Architektur und folgte dem Hardware-Paradigma der EDVAC, aber basierte auf einer anderen Form des Befehlsformats, ohne konventionelle Opcodes. Wie Campbell-Kelly bemerkt, sind «die meisten Computer einander hinreichend ähnlich, um es einem sachkundigen Programmierer aufgrund des Befehlsformats und einer Tabelle von Opcodes zu gestatten, ein angemessenes Verständnis der Maschine zu erlangen. Der Erbauer der ACE ist dabei eine Ausnahme, weil seine Architektur der anderer moderner Computer ziemlich unähnlich blieb […]».[24] Nach ihrem Umbau im Jahre 1948 folgte die ENIAC dem modernen Code-Paradigma, und zwar mit überraschender Treue. Das Nutzungsgefühl und die Struktur des Codes zeigt eine unverkennbare Verwandtschaft mit denen anderer früher Maschinen.

Die Maschinen, die Mitte der 1950er Jahre gebaut wurden, implementierten meist alle drei paradigmatischen Aspekte der «First-Draft»-Konstruktion der EDVAC. Gegen Ende des Jahrzehnts änderten sich die jeweiligen Positionen dieser drei Paradigmen wieder. Die Bedeutung des Hardware-Paradigmas verblasste als erste, und zwar als Transistoren und Kernspeicher Elektronenröhren und Verzögerungsleitungen überflüssig machten. Das Paradigma der Von-Neumann-Architektur genoss ein längeres Leben, obwohl seine Vorrangstellung nach und nach unterminiert wurde, als Innovationen wie Parallelverarbeitung, Message Passing Interface, Befehls-Pipeline, Speicherdirektzugriff durch Peripheriegeräte, Stapelspeicher und Memory Address Register Stück für Stück ihren radikalen Minimalismus auflösten.

Das moderne Code-Paradigma hingegen blieb als Beschreibung der vom Prozessor ausgeführten Maschinensprache (wenngleich auch nicht der Sprachen, die Menschen nutzen, um Programme zu schreiben) größtenteils intakt. Es wurde erweitert und in vielerlei Hinsicht präzisiert, nicht zuletzt durch von Neumanns eigene Beschreibung der geplanten Struktur seiner *Institute for Advanced Studies Machine*.[25] Gekippt wurde es allerdings nie.

Aus dem Englischen von Johannes Paßmann

23 Martin Campbell-Kelly: *Foundations of Computer Programming in Britain (1945–1955)*, Dissertation, Sunderland Polytechnic 1980, 239.

24 Martin Campbell-Kelly: Programming the Pilot Ace. Early Programming Activity at the National Physical Laboratory, in: IEEE *Annals of the History of Computing*, Jg. 3, Nr. 2, 1981, 133–162, hier 138.

25 Arthur W. Burks, Herman H. Goldstine, John von Neumann: *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument*, Princeton, New Jersey 1946.