

Abschnitt 1intr

17. Juni 2000

Dieses Werk ist urheberrechtlich geschützt; alle Rechte mit Ausnahme des folgenden sind vorbehalten:

Studenten dürfen für ihre persönlichen Lernzwecke dieses Dokument ausdrucken und auf ihren Rechnern Kopien der entsprechenden Datei vorhalten.

Ausdrücklich verboten wird jede andere Verwendung von Ausdrucken und Dateikopien; insbesondere darf dieses Skriptum nicht in irgendeiner Weise vertrieben werden und es dürfen keine Kopien der Datei auf irgendwelchen „Skriptenservern“ angelegt werden. Gegen die Anlage von Verweisen („hypertext links“) auf diese Datei ist selbstverständlich nichts einzuwenden.

Copyright © Herbert Klaeren, 1999.

Inhaltsverzeichnis

1	Einführung	9
1.1	Was ist Informatik?	9
1.2	Geschichte der Programmierung	17
2	Induktive Definitionen	21
2.1	Natürliche Zahlen	21
2.2	Wortmengen	28
2.3	Syntaktische Beschreibungsmittel	32
2.4	Terme	40
2.5	Aufgaben	47
3	Algorithmen und Programme	49
3.1	Algorithmus	49
3.2	Programme	54
3.2.1	Ausdrücke, Namen	55
3.2.2	Fallunterscheidungen	59
3.2.3	Rekursive Definitionen	60
3.3	Aufgaben	62
4	Abstrakte Datentypen	93
4.1	Einführung	93
4.2	Boolesche Werte	95
4.3	Zähler	98
4.4	Listen	101
4.5	Bäume	108
4.6	Aufgaben	116
5	Logische Kalküle	117
5.1	Ein Kalkül für die Aussagenlogik	118
5.2	Ein Kalkül für die Prädikatenlogik	122
5.3	Der Reduktionskalkül RC_1	125
5.4	Der λ -Kalkül	127
A	Mathematische Grundlagen	119
A.1	Mengen, Relationen, Abbildungen	119

A.2	Formale Logik	124
	A.2.1 Aussagenlogik	124
	A.2.2 Prädikatenlogik	126
A.3	Halbordnungen	128
A.4	Aufgaben	129

Kapitel 1

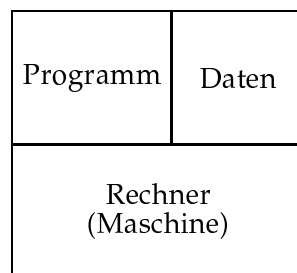
Einführung

1.1 Was ist Informatik?

Seit etwa 1967 gibt es in der Bundesrepublik den Studiengang Informatik. In Amerika ist das entsprechende Wort *informatics* nicht sehr verbreitet, dort sind die Bezeichnungen „*Computer Science*“, „*Computing Science(s)*“ und „*Information Science(s)*“ üblich. Mit diesen unterschiedlichen Bezeichnungen ist natürlich auch immer eine unterschiedliche Sicht der Informatik verbunden. Die deutsche Bezeichnung wurde damals in Anlehnung an den von der *Académie Française* geprägten Begriff „*informatique*“ gewählt, die dort als „*traitement rationnel de l'information*“ (rationale Behandlung der Information) definiert wurde.

Der Gleichklang von „Informatik“ mit „Mathematik“ ist sicher mehr als zufällig. Die Wortschöpfer wollten deutlich machen, daß es in der Informatik um mehr geht als bloß um Computer. Gegenstand sind vielmehr *mathematische Modelle zur Informationsverarbeitung*, die Computergenerationen überdauern.

Damit ein *Rechenprozeß* oder *Informationsverarbeitungsprozeß* ablaufen kann, sind drei Dinge vonnöten:



Die Informatik beschäftigt sich deshalb mit

- Struktur, Wirkungsweise, Fähigkeiten und Konstruktionsprinzipien

von Informationsverarbeitungssystemen,

- Strukturen, Eigenschaften und Beschreibungsmöglichkeiten von *Informationen* und von Informationsverarbeitungsprozessen,
- Möglichkeiten der Strukturierung, Formalisierung und Mathematisierung von *Anwendungsgebieten* sowie der Modellbildung und Simulation.

Die Durchführung irgendeiner Tätigkeit auf dem Computer erfordert eine Formalisierung dieser Tätigkeit bis in die feinsten Details; daher werden auch in der Informatik stets *formale Methoden* eingesetzt. Das heißt, wir beschäftigen uns mit

- abstrakten Zeichen, Objekten und Begriffen,
- formalen Strukturen (z.B. Sprachstrukturen, Datenstrukturen) und ihrer Transformation nach formalen Regeln,
- der effektiven Darstellung solcher Strukturen im Rechner und der automatischen Durchführung der Transformationen.

Damit hat die Informatik auch Züge einer Ingenieurswissenschaft. In der ENCYCLOPAEDIA BRITANNICA wird die Tätigkeit eines Ingenieurs als

„die schöpferische Anwendung wissenschaftlicher Prinzipien auf Entwurf und Entwicklung von Strukturen, Maschinen, Apparaten oder Herstellungsprozessen oder Arbeiten, wobei diese einzeln oder in Kombination verwendet werden; oder dies alles zu konstruieren und zu betreiben in voller Kenntnis seines Entwurfs; oder dessen Verhalten unter bestimmten Betriebsbedingungen vorherzusagen; alles dies im Hinblick auf eine gewünschte Funktion, Wirtschaftlichkeit des Betriebs und Sicherheit von Leben und Eigentum“

definiert. Hieraus sollen die drei folgenden Punkte hervorgehoben werden, die den Charakter der Informatik als Ingenieurswissenschaft betonen:

1. Erstes Ziel ist die Erbringung einer gewünschten *Funktion*, das heißt also der Schritt von einem Problem zu einer Lösung. Die meisten lösbaren Probleme haben unübersehbar viele Lösungen. Um diese aufzufinden, ist in der Regel eine breite Kenntnis von Grundlagen und

Methoden, aber auch Kreativität erforderlich. Es gehört zum Handwerk des Ingenieurs, nachzuweisen, daß die gewünschte Funktion tatsächlich erbracht wird. Vieles an diesem Nachweis wird mathematischer Natur sein (vgl. etwa die Baustatik).

2. Im Gegensatz zur Mathematik, wo man sich vielfach mit reinen Existenzbeweisen zufriedengibt, welche ein Problem „im Prinzip“ lösen, muß der Informatiker wie auch der Ingenieur darauf dringen, daß seine Problemlösung wirtschaftlich vertretbar („*effizient*“) ist. Unter allen theoretisch denkbaren Lösungsmöglichkeiten ist daher möglichst die kostengünstigste auszuwählen.
3. *Sicherheitsbetrachtungen* gehören von jeher zu den Pflichtübungen des Ingenieurs. Der bestimmungsgemäße Gebrauch eines Geräts darf weder den Benutzer, noch den Rest des Systems gefährden, von dem diese Vorrichtung ein Teil ist. Das erfordert eine sehr sorgfältige Analyse der Grenzfälle des Betriebs. Es gehört zum guten Ton, (bis zu einem gewissen Grad) auch den nicht bestimmungsgemäßen Gebrauch entweder konstruktiv zu verhindern oder aber doch möglichst sicher zu gestalten. In jedem Fall versucht der Ingenieur, durch gewisse Sicherungen einen denkbaren Schaden möglichst gering zu halten.

Informationen kann man nicht losgelöst von der Welt betrachten, in der sie etwas bedeuten. Informatiker¹ erarbeiten Lösungen für Probleme von Menschen und Organisationen. Sie müssen immer interdisziplinär arbeiten und einen multiperspektivischen Ansatz verfolgen. Das macht die Arbeit der Informatiker so spannend. Wir haben es in der Regel nicht mit klar definierten Aufgaben zu tun, die wir „nur“ lösen müssen, sondern meist müssen wir die eigentliche Aufgabe erst herausfinden.

Informatik ist eine sehr *formale Wissenschaft*. Dies ist nicht zuletzt dadurch bedingt, daß Computerprogramme sehr formale Objekte sind; selbst kleinste Unterschiede wie die Ersetzung eines Kommas durch einen Punkt können hier schwerwiegende Konsequenzen haben. Aber auch schon bevor wir Programme schreiben, müssen wir in der Regel sehr formal (d. h. mathematisch) arbeiten. Im Anhang A stellen wir einige mathematische Grundlagen relativ knapp dar; weitergehender Bedarf kann durch das sehr gute Lehrbuch von Knuth et al. [GKP89] abgedeckt werden.

¹mit denen immer auch die Informatikerinnen angesprochen sind

Die Anwendungswelt, innerhalb derer sich die Informatikprodukte bewähren müssen, ist in aller Regel aber nicht formalisiert; es ist eine *informale Welt*. Die Informatiker müssen immer zwischen der formalen und der informalen Welt vermitteln. Ihre kommunikativen Fähigkeiten sind dabei stark gefordert.

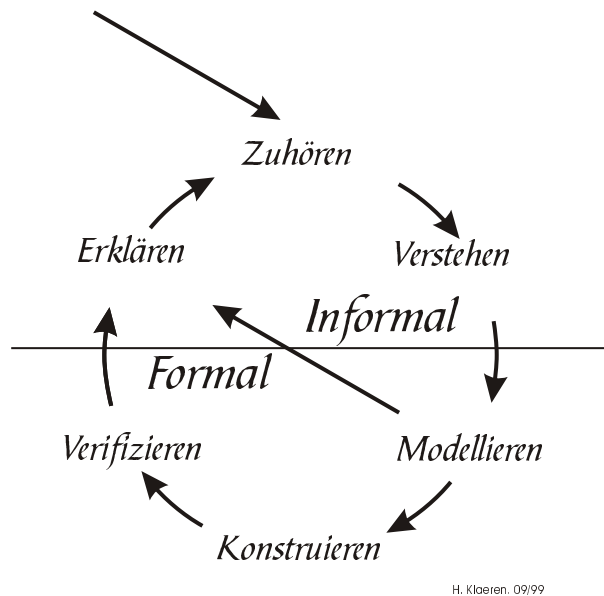


Abbildung 1.1: Entwicklung eines Informatikprodukts

In Abb. 1.1 ist das typische Arbeitsverfahren bei der Herstellung von Informatikprodukten schematisch dargestellt; die Linie in der Mitte trennt die informale von der formalen Welt. Wir beginnen in der informalen Welt, indem wir einem zukünftigen Benutzer unseres Produkts („Auftraggeber“) aufmerksam zuhören und verstehen, welche Anforderungen er an das Produkt hat. Gleich nach dem Verstehen gehen wir in die formale Welt über, indem wir uns ein *Modell* des geplanten Produkts herstellen. Natürlich wird der Auftraggeber, der in der informalen Welt zu Hause ist, das Ergebnis der Modellierung nicht verstehen können. Wir müssen ihm deshalb die Eigenschaften unseres Modells, d. h. also das, was wir bis jetzt verstanden

und formal beschrieben haben, erklären. Möglicherweise wird sich dabei herausstellen, daß wir etwas falsch oder noch nicht vollständig verstanden haben, oder durch unsere Darstellung wird der Auftraggeber angeleitet, seine Vorstellungen zu erweitern oder zu revidieren. Nach einigen Iterationen des kurzen Zyklus „Zuhören–Verstehen–Modellieren–Erklären“ werden wir das Gefühl haben, die Aufgabe nun gründlich verstanden zu haben. Wir beginnen dann damit, das Produkt zu konstruieren (selbstverständlich in der formalen Welt) und anschließend für uns selbst zu verifizieren, daß dieses Produkt unserem Modell entspricht. Anschließend müssen wir das fertige Produkt dem Auftraggeber wiederum erklären, was möglicherweise weitere Mißverständnisse und Änderungswünsche zutage fördert.

Wir sind alle mit dem Lösen von *Textaufgaben* oder *Denksportaufgaben* vertraut, die wir hier vielleicht einmal als Prototyp von *informellen Problembeschreibungen* auffassen können. Wir müssen dabei jeweils zuerst feststellen, was wirklich zu tun ist. Die Informatiker sprechen von der *Spezifikation* des Problems. Speziell die Denksportaufgaben zeigen, daß häufig die eigentliche Leistung in der Aufstellung der Spezifikation liegt: Wenn erst einmal klar ist, was zu rechnen ist, ist die Aufgabe schon zu 90% gelöst.

An einem ganz banalen Beispiel wollen wir die Tücken informeller Problembeschreibungen aufzeigen:

Beispiel 1.1 Auf einem Parkplatz stehen PKW's und Motorräder ohne Beiwagen. Zusammen seien es n Fahrzeuge mit insgesamt m Rädern. Bestimme die Anzahl P der PKW's.

Bevor wir mit einer genaueren Betrachtung dieses Problems beginnen, beobachten wir, daß hier in der Tat eine ganze *Klasse von Problemen* beschrieben, nämlich je ein gesondertes Problem für jede mögliche Wahl von n und m . Wir sagen auch: Die Problembeschreibung enthält n und m als *Parameter*. Das Problem ist offensichtlich die Berechnung eines Funktionswerts $P(n, m)$.

Wir überlegen, daß offensichtlich die Anzahl $P(n, m)$ der PKW's plus die Anzahl $M(n, m)$ der Motorräder die Gesamtzahl n der Fahrzeuge ergibt. Außerdem wissen wir, daß jeder PKW 4 Räder und jedes Motorrad 2 Räder hat und daß die Radzahlen der PKW's und Motorräder zusammen m ergeben müssen. Wenn wir also für $P(n, m)$ einfach P und für $M(n, m)$

einfach M schreiben, haben wir es mit dem folgenden Gleichungssystem zu tun:

$$\begin{aligned} P + M &= n \\ 4P + 2M &= m \end{aligned}$$

Wir lösen die erste Gleichung nach M auf:

$$M = n - P$$

und setzen in die zweite Gleichung ein:

$$\begin{aligned} 4P + 2(n - P) &= m \\ 2P &= m - 2n \\ P &= \frac{m - 2n}{2} \end{aligned}$$

An dieser Stelle sind wir versucht, das Problem als gelöst zu betrachten und könnten jetzt nach dieser Formel ein Computerprogramm schreiben. Auf diese Weise sind auch die berühmten „Null-Mark-Rechnungen“ des Computers entstanden, die dann auch noch von der „Null-Mark-Mahnung“ gefolgt werden und schließlich nur durch eine fiktive „Null-Mark-Überweisung“ abgestellt werden können. Wir erhalten nämlich für $n = 3$ und $m = 9$

$$P(3, 9) = \frac{9 - 2 \cdot 3}{2} = \frac{3}{2} = 1,5 \quad ,$$

also müßten auf dem Parkplatz anderthalb PKW's stehen. Offensichtlich ergibt die Aufgabe nur einen Sinn, wenn die Anzahl m der Räder gerade ist. Aber das ist noch nicht alles, wie die Rechnung für $n = 5$ und $m = 2$ zeigt:

$$P(5, 2) = \frac{2 - 2 \cdot 5}{2} = \frac{2(1 - 5)}{2} = 1 - 5 = -4$$

Die Antwort wäre also „*Es fehlen vier PKW's*“, was auch unsinnig ist. Die Anzahl der Räder muß mindestens zweimal so groß sein wie die Anzahl der Fahrzeuge. Machen wir noch einen dritten Versuch mit $n = 2$ und $m = 10$:

$$P(2, 10) = \frac{10 - 2 \cdot 2}{2} = \frac{10 - 4}{2} = 3$$

Dieses Beispiel ist sogar noch schlimmer als die anderen beiden, denn hier ist auf den ersten Blick nicht so deutlich ersichtlich, daß das Ergebnis unsinnig ist. Erst eine zweite Überlegung erhebt die Frage, wieso eigentlich drei von zwei Fahrzeugen PKWs sein können. Die Antwort ist in diesem Fall, daß das dritte Fahrzeug ein „negatives Motorrad“ ist. In Wirklichkeit kann nämlich die Anzahl der Räder höchstens viermal so groß sein wie die Anzahl der Fahrzeuge.

Der „Fehler“ in der Problembeschreibung dieses Beispiels ist, daß einige Tatsachen aus der Anschauungs- und Erfahrungswelt als allgemein bekannt vorausgesetzt werden, die wir in der obigen Überlegung auch explizit angesprochen haben. Unter anderem wird davon ausgegangen, daß es sich bei n und m tatsächlich um die Fahrzeugzahl und Räderzahl eines real existierenden Parkplatzes handelt; dann können die hier angesprochenen Probleme natürlich nicht auftreten. Für die mathematische Behandlung in der oben abgeleiteten Formel sind jedoch n und m nur irgendwelche Zahlen; der Bezug zu real existierenden Gegenständen ist völlig aufgehoben. Derartige *Abstraktionsschritte* sind typisch für die Mathematik, aber auch für die Informatik. In der Informatik allerdings sind die Folgen einer inkonsequenten Einhaltung von Abstraktionen in der Regel verheerender, weil sie durch die Computer vervielfacht werden.

Ein vorsichtiger Informatiker würde deshalb für das Parkplatzproblem zunächst eine *Spezifikation* schreiben, die etwa wie folgt aussehen könnte:

Spezifikation 1.2 (Parkplatzproblem) Zu bestimmen ist die Anzahl $p(n, m)$ der PKW's auf einem Parkplatz mit n Fahrzeugen und zusammen m Rädern. Die Fahrzeuge sind nur PKW's und Motorräder.

Eingabe: $m, n \in \mathbb{N}$

Vorbedingung: m gerade, $2n \leq m \leq 4n$

Ausgabe: $p(n, m) \stackrel{\text{def}}{=} P \in \mathbb{N}$, falls Nachbedingung erfüllbar, sonst „Keine Lösung“.

Nachbedingung: Für gewisse $P, M \in \mathbb{N}$ gilt

$$\begin{aligned} P + M &= n \\ 4P + 2M &= m \end{aligned}$$

Eine solche *funktionale Spezifikation* beschreibt die Menge der gültigen Eingabegrößen („*Definitionsbereich*“) und die Menge der möglichen Ausgabegrößen („*Wertebereich*“) mit allen für die Lösung wichtigen Eigenschaften, insbesondere dem funktionalen Zusammenhang zwischen ihnen. Diesen Zusammenhang beschreibt man gerne durch sogenannte *Vor-* und *Nachbedingungen*². Die *Vorbedingung* beschreibt den Zustand *vor* Ausführung des geplanten Algorithmus, m.a.W. seine *Voraussetzungen*. Die *Nachbedingung* beschreibt dementsprechend den Zustand *nach* Ausführung des Algorithmus, m.a.W. seine *Leistungen*.

In dieser Spezifikation sind wir extrem vorsichtig vorgegangen, indem wir auch den Fall vorgesehen haben, daß das angegebene Gleichungssystem nicht über den natürlichen Zahlen lösbar ist. In der Praxis hat man es in der Tat häufiger mit der Berechnung von *partiellen Abbildungen* zu tun, bei denen nicht zu allen Parametern Werte existieren. Deshalb kann die Nachbedingung auch nicht in allen Fällen erfüllt werden. Wir haben hier die offensichtlichsten Voraussetzungen unter „*Vorbedingung*“ aufgeführt. Natürlich hätte man auch noch hinzufügen können: „Es existieren $P, M \in \mathbb{N}$ so daß die Nachbedingung erfüllt ist.“ Damit hätte man allerdings auf unschöne Art *Vor-* und *Nachbedingung* miteinander verknüpft bzw. die *Nachbedingung* bereits vorweggenommen. Außerdem ließe sich die *Nachbedingung* erst überprüfen, wenn wir bereits versucht haben, das Ergebnis zu ermitteln. Häufig verlangt man jedoch von Algorithmen (und Programmen), daß sie *robust* sind, d.h. die Einhaltung ihrer *Vorbedingung* selbst überprüfen, bevor sie mit der Rechnung beginnen. In unserem Beispiel kann man sofort einsehen, daß die *Vorbedingung* hinreichend für die Lösbarkeit des Gleichungssystems über \mathbb{N} ist: Wenn die Zahl der Räder m gerade ist, dann ist auch $m - 2n$ gerade, so daß die Zahl der PKW's P in jedem Fall eine ganze Zahl wird. Die Bedingung $2n \leq m$ sorgt dafür, daß P nicht negativ wird und die Bedingung $m \leq 4n$ dafür, daß der maximale Wert von P gleich n ist; dadurch werden negative Motorradzahlen vermieden.

²auch *Anforderungen* und *Zusicherungen*

1.2 Geschichte der Programmierung

Wir können hier keinen vollständigen Überblick über die Geschichte der Programmierung, geschweige denn der Informatik, geben; wer sich dafür interessiert, möge etwa in [MHR80, Wex81] nachsehen. Hier referieren wir nur kurz einige Punkte, die auch erkennen lassen, wie die Informatik als Wissenschaft entstanden ist.

Die Geschichte der Programmierung ist länger als die Geschichte der Computer. Wir haben bereits zuvor auf die Kombination *Maschine* \leftrightarrow *Programm* hingewiesen, die besonders für Computer charakteristisch ist. Aber auch andere Maschinen sind oder waren mehr oder weniger programmierbar:

- Bereits JOSEPH JACQUARD (1752–1834) stellte einen Webstuhl vor, der mit Hilfe von *Lochkarten* auf die Herstellung beliebig gemusterter Stoffe programmiert werden konnte.
- HERMANN HOLLERITH (1860–1929) entwarf, angeregt durch die Jacquard'schen Lochkarten Kartenlocher, Sortierer und Zähler zur Auswertung der amerikanischen Volkszählung von 1890. Diese Maschinen waren durch *Steckbretter* programmierbar, wobei man mit Hilfe von kurzen Schnüren Abtaster für Eingabelochkarten mit Zählwerken, Addierern, Stanzmagneten für Ausgabelochkarten etc. verband. Nach diesem Prinzip funktionierende sogenannte „*Rechenstanzer*“ waren bei uns noch Ende der Sechziger Jahre in Betrieb.
- Werkzeugmaschinen (sog. NC-Maschinen) werden bereits sehr lange durch *Lochstreifen* auf bestimmte Bearbeitungsgänge programmiert .

Auch die ersten echten Computer waren sogenannte *Fixed Program Machines*, d.h. also Maschinen, in denen das Programm durch mechanische Manipulationen fest montiert wurde. Bei der Z3 zum Beispiel, die von dem deutschen Computer-Pionier KONRAD ZUSE 1941 gebaut wurde, war das Programm auf einem Lochstreifen enthalten, den die Maschine Schritt für Schritt abarbeitete. Wenn der Algorithmus es erforderte, daß bestimmte Prozesse häufiger wiederholt werden mußten, bis ein *Abbruchkriterium* erfüllt war, klebte Zuse den entsprechenden Lochstreifen zu einem Ring bzw. einer *Schleife* zusammen. Interessanterweise sprechen wir heute auch im Zusammenhang mit *Wiederholungsanweisungen* von Schleifen.

Beim *ENIAC* (Electronic Numerical Integrator And Computer), der 1943 in den USA unter Leitung von J.P. ECKERT und J. MAUCHLY als erster voll-elektronischer Computer entstand und deshalb gelegentlich fälschlicherweise als *der* erste Computer bezeichnet wird, waren die Programme in Steckbrettern ähnlich denen der Rechenstanzer enthalten. Im Prinzip bestand ein Programm darin, daß man an sich autonome Maschinenteile wie Register, Addierer, Zähler, Schrittfolge-Generatoren etc. untereinander verband und damit den mathematischen Algorithmus nachbildete, der hier zu berechnen war. Die Denkweise, die durch diese Maschinen erzwungen wurde, das sogenannte *Datenflußdenken*, ist heute wieder sehr modern.

In der Tat dachten die Konstrukteure der ersten Computer zunächst nicht daran, daß diese Maschinen in einem gewissen Sinne *universell* einsetzbar waren; vielmehr stand die Lösung ganz bestimmter Probleme im Vordergrund. Diese ersten Computer wurden in der Regel von ihren Erbauern programmiert; diese waren zugleich auch die Benutzer der Rechner. Insofern gab es wenig Gelegenheit für Mißverständnisse; darüber hinaus hatten die von den Programmen gelieferten Ergebnisse keine unmittelbaren Auswirkungen auf die reale Welt. Außerdem standen die Algorithmen, d.h. also die Rechenvorschriften bereits weitgehend fest und mußten lediglich noch codiert werden. In dieser Umgebung war natürlich kein Bedarf für eine neue Wissenschaft *Informatik* oder *Computer Science*; die Methoden der Mathematik und der Elektrotechnik waren völlig ausreichend.

Fassen wir nochmals zusammen, was für die Frühzeit der Computer (etwa 1942–1948) galt:

- *Fixed Program Machines*: Das Programm ist nicht in der Maschine gespeichert, sondern stellt eine Art Schaltung dar, die den Datenfluß eines mathematischen Algorithmus nachbildet.
- Die Algorithmen und ihre Eigenschaften sind aus der Mathematik bekannt.
- Die Konstrukteure sind gleichzeitig auch die Programmierer und die Benutzer.
- Die Programme behandeln eine eingeschränkte Problemklasse und haben keine unmittelbaren praktischen Auswirkungen.

Gegen Ende der Vierziger Jahre kamen (unter dem Einfluß einer billigeren Speichertechnologie) die ersten Maschinen auf, in denen das Programm

in gleicher Weise gespeichert war wie zuvor nur die Daten. Die Idee, daß Programme auch Daten sind und von der Maschine unter der Kontrolle anderer Programme manipuliert werden können, wird meistens dem Mathematiker JOHN VON NEUMANN zugeschrieben, obwohl mit Sicherheit auch andere zu dieser Zeit daran gedacht haben [MHR80].

Die Auswirkungen dieser Idee sind schier unermesslich: Es wird jetzt möglich, daß der Computer selbst seine Programme herstellt, wobei der Mensch „nur noch“ zu *spezifizieren* braucht, was denn zu tun ist. Dies kann er auf einer höheren Ebene tun als zuvor; er braucht nicht mehr die in der Mathematik gebräuchlichen Variablen selbst auf die Maschinenregister abzubilden und braucht sich nicht mehr so intensiv mit der Hardware der Maschine auseinanderzusetzen. Bereits 1952/53 erschienen die ersten *Programmgeneratoren*, die es erlaubten, daß man arithmetische Ausdrücke wie z.B. „ $2 + x * (i - 5)$ “ fast wie in der Alltagsmathematik aufschreiben konnte; diese wurden automatisch in eine Folge von Befehlen zur Berechnung dieses Ausdrucks übersetzt.

Bis heute sind mehrere hundert Programmiersprachen entwickelt worden, was sicherlich dazu veranlaßt, daß man sich über Programmierung und Programmiersprachen im allgemeinen Gedanken machen sollte, da man sowieso niemals alle Programmiersprachen im Einzelnen erlernen wird.

Wir haben zuvor vier wesentliche Merkmale der Frühzeit der Programmierung vorgestellt; stellen wir jetzt dem gegenüber, was danach charakteristisch wurde und bis heute gilt:

- *Stored Program Machines*: Das Programm ist selbst in der Maschine gespeichert. Es wird (normalerweise) nicht durch mechanische Manipulationen in die Maschine eingebracht, sondern unter Kontrolle anderer Programme (*Betriebssysteme*) eingelesen und gestartet. Programme werden vom Programmierer nicht in der Codierung erstellt, wie sie die Maschine benötigt, sondern in einer *Programmiersprache* geschrieben.
- In den meisten Fällen ist der Computer nur Teil einer ganzheitlichen Problemlösung; die genaue Aufteilung der Aufgabe sowie geeignete Algorithmen zu ihrer maschinellen Lösung müssen erst gesucht werden.
- Maschinenhersteller, Programmierer und Benutzer der Rechner bzw.

Programme sind verschiedene Menschengruppen mit unterschiedlicher Vorbildung. Dies hat Kommunikationsprobleme zur Folge und schafft die Notwendigkeit von Dokumentationen.

- Der Einsatzbereich der Programme ist unüberschaubar; die Programme haben z.T. unmittelbare praktische Auswirkungen.

Die hier auftretenden Probleme verlangten in der Tat nach der Informatik als einer neuen Wissenschaft, die sich damit ganzheitlich beschäftigen konnte. Maschinen mit einem gespeicherten Programm können unter Kontrolle unterschiedlicher Programme ganz unterschiedliche Leistungen erbringen. Wenn wir annehmen, daß der Rechner unsere Eingabesprache „unmittelbar versteht“, bilden wir damit *scheinbare (virtuelle) Maschinen*. Vieles von dem, was als Leistung der Maschine selbst erscheint, ist jedoch Leistung eines Programms, und die Verwendung von sog. „Nur-Lese-Speichern“ (ROM's) verwischt die Grenzen zwischen Hardware und Software.

Der geschickte Entwurf einer Hierarchie von virtuellen Maschinen und die effiziente Übersetzung zwischen ihren Sprachen ist eine der wesentlichen Herausforderungen bei der Programmierung. In modernen Ansätzen des „Hardware-Software-Codesigns“ wird für spezialisierte Anwendungen sogar die Hardware parallel zur Software entwickelt.

Literaturverzeichnis

- [Bau68] BAUMANN, RICHARD: *Algol-Manual der Alcor-Gruppe*. R. Oldenbourg, München/Wien, 3. Auflage, 1968.
- [Bau89] BAUER, F. L.: *100 Jahre Peano-Zahlen*. Informatik-Spektrum, 12:340–341, 1989.
- [End72] ENDERTON, H. B.: *A Mathematical Introduction to Logic*. Academic Press, 1972.
- [GKP89] GRAHAM, R. L., D. E. KNUTH und O. PATASHNIK: *Concrete Mathematics*. Addison-Wesley, 1989.
- [Hal69] HALMOS, P.: *Naive Mengenlehre*. Vandenhoeck & Ruprecht, Göttingen, 1969.
- [KCR98] KELSEY, RICHARD, WILLIAM CLINGER und JONATHAN REES: *Revised⁵ Report on the Algorithmic Language Scheme*. SIGPLAN Notices, 33(9):26–76, 1998.
- [Kla83] KLAEREN, HERBERT: *Algebraische Spezifikation — Eine Einführung*. Springer Verlag, Berlin-Heidelberg-New York, 1983.
- [Knu73] KNUTH, D. E.: *The Art of Computer Programming*, volume 3. Addison-Wesley, 1973. Sorting and Searching.
- [Mes71] MESCHKOWSKI, H.: *Einführung in die moderne Mathematik*, Band 75/75a der Reihe *Hochschultaschenbücher*. BI, 1971.
- [MHR80] METROPOLIS, N., J. HOWLETT, and GIAN-CARLO ROTA (editors): *A History of Computing in the Twentieth Century*. Academic Press, 1980.
- [Rie91] RIESE, ADAM: *Rechnung auff der Linihen und Federn /Auff allerley handthirung gemacht / durch Adam Risen (Faksimile 2. Aufl. Erfurt 1532)*. Magistrat der Stadt Erfurt, 1991.
- [Wex81] WEXELBLAT, RICHARD L. (editor): *History of Programming Languages*, New York, 1981. Academic Press.