In dem folgenden dritten Teil ihrer Serie entwickeln Clapes, Lynch und Steinberg die Grundlagen für den zentralen Teil ihrer These: Die Programmier-„Sprachen" sind wirkliche, lesbare Sprachen wie die natürlichen Sprachen auch, die wie diese das Ziel verfolgen, (auch) von Menschen gelesen zu werden. Deshalb gilt nach Ansicht der Autoren zweierlei:
Die auf diesen Punkt abzielende Beweisführung präsentiert sich zugleich wieder als ein Grundkurs in Sachen Computertechnik, heute zum Thema „Programmiersprachen", wobei auch die Geschichte dieser Kreationen nicht zu kurz kommt. Am Ende weiß man dann, was es mit Maschinensprache, Assembler und höheren Programmiersprachen auf sich hat und kann kompetent prüfen, ob man den von den Autoren gezogenen Schlußfolgerungen zustimmen will.

Das Epos von Silicon und die Barden des Binären

Zur Bestimmung der korrekten Reichweite des urheberrechtlichen Schutzes für Computer-Software

Teil 3

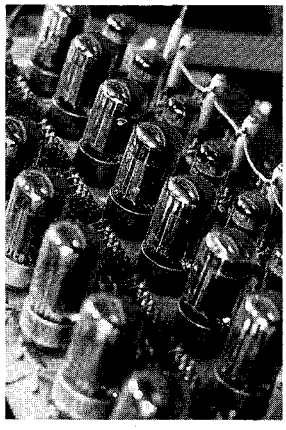
Anthony L. Clapes
Patrick Lynch
Mark R. Steinberg

III. Die Entscheidung der Kontroverse

A. Wie Programmierer sich ausdrücken

2. Der Kontext, in dem Programme geschrieben werden

b. Die Sprache(n) der Computer



Die bisherige Diskussion zeigt, daß die Computer-Hardware der Art und Weise, in der Programme geschrieben werden können, bestimmte Anforderungen auferlegt. Diese Anforderungen jedoch betreffen eine Sicht des Programmierens auf der primitivsten Stufe, gewissermaßen auf dem Niveau einer Ursprache, die beim gegenwärtigen Sprechen nicht mehr verwandt wird. Im Unterschied dazu haben Programmierer Sprachen entwickelt, die der menschlichen Ausdrucksweise näher stehen. Aus diesen Sprachen übersetzen Computer dann in die elementaren Muster von 1 und 0. Wir werden jetzt einen kurzen Blick auf die Entwicklung solcher Sprachen im Laufe der Geschichte der Computeranwendungen werfen.
Einer der ersten programmierbaren Computer war der ENIAC, der 1946 in der Universität von Pennsylvania entworfen und gebaut worden war.[82] Er wog 30 Tonnen, enthielt 17000 Röhren und nahm 1800 Quadratfuß in Anspruch.[83]
Der ENIAC war dafür konzipiert, die Flugbahn von Artilleriegeschossen zu berechnen. Er konnte aber auch für die Lösung anderer Probleme vorbereitet werden. Für diese Anpassung mußte der Operator des ENIAC einen Teil der 6000 Schalter in der Maschine umstellen und einen Teil der Hunderte von Kabelverbindungen umstecken, ähnlich wie Telefon-Vermittler bei den altmodischen Telefonvermittlungen. Tatsächlich verdrahtete man auf diese Weise die Verbindungen zwischen den ENIAC-Schaltkreisen neu. Zweifelsohne benutzten die Techniker, die diese Arbeit vornahmen, schriftliche Instruktionen der Maschinen-Planer, die in einer für Menschen lesbaren Form in Dia-

Der Beitrag ist mit dem Titel „Silicon Epics and Binary Bards: Determining the Proper Scope of Copyright for Computer Programs" zuerst im UCLA Law Review (vol. 34, no. 5 & 6) erschienen. Wir danken für die freundliche Erlaubnis, ihn hier in deutscher Übersetzung bringen zu dürfen.
– Anthony L. Clapes ist „Senior Corporate Counsel" bei IBM (Armonk, N.Y.), Patrick Lynch und Mark R. Steinberg sind Partner der Kanzlei Melveny and Myers (Los Angeles, CA).

82 Die Entwicklung des ENIAC ist beschrieben bei S. Augarten, Bit by Bit: An Illustrated History of Computers and their Inventors 107-31 (1984). Eine gewisse Kontroverse begleitet die Geschichte dieses Entwicklungsprojekts und die von dessen Nachfolger (dem EDVAC). Die Beteiligten sind tiefgreifend darüber uneinig, wer für welche Aspekte der Entwicklung verantwortlich zeichnete. Drei persönliche Ansichten dieser Art sind zu finden bei H. Goldstine, The Computer from Pascal to Neumann 149–84 (1972); H. Lukoff, From Dits to Bits: A Personal History of the Electronic Computer 29–41 (1979); und N. Stern, From ENIAC to UNIVAC 7–86 (1981). Die Beschreibung des ENIAC-Computers im Text und die Darstellung des Konzepts „gespeichertes Programm", das aus den Programmierschwierigkeiten beim ENIAC erwuchs, beruhen auf S. Augarten, a.a.O., bei 121–42 und N. Stern, a.a.O., bei 50–51.

83 Heute sind die elektronischen Taschenrechner leistungsfähiger.

grammen oder sprachlichen Darstellungen angaben, welche Schalter umgelegt werden sollten und wie die Kabel gesteckt werden sollten, um die Maschine zur Herstellung des gewünschten Resultats zu veranlassen.

Statt „Neuverdrahten" des Rechners ...

Das Neu-Verdrahten des Computers war offensichtlich ein zeitaufwendiger fehleranfälliger Prozeß. Wäre dieser Prozeß nicht mechanisiert worden, so gäbe es heute keine kommerzielle Computerindustrie. Glücklicherweise war eine der Persönlichkeiten, die mit dem ENIAC-Projekt vertraut waren, der große Mathematiker John von Neumann. Von Neumann formulierte als erster die Idee, daß die Instruktionen für das Umstecken des ENIAC in sich eine Art von Information seien, die ebenso wie die Daten, über die der Computer operierte, im Computer gespeichert werden könne. Durch die Abarbeitung der Instruktionen (heute als „Programm" bezeichnet) durch den Computer zusammen mit den Daten und durch die gleichzeitige Information an den Computer, durch welche seiner Schaltkreise die Daten fließen sollten und in welcher Abfolge, konnte die Aufgabe des Neuverdrahtens des Computers ausgeschaltet werden. Der Computer wurde auf diese Weise wirklich zu einem Mehrzweck-Problemlösungsinstrument.

... ein „gespeichertes Programm"

Dieses Konzept des „gespeicherten Programms" ist der Schlüssel für die industrielle Entwicklung und für das Verständnis des Computers. Indem man (beispielsweise auf Diskette oder Magnetband) eine breite Vielzahl von Programmen speichert und sie dann je nach Bedarf in den Hauptspeicher lädt, kann derselbe Computer dazu veranlaßt werden, eine Gehaltsabrechnung durchzuführen, eine Kundengeschichte aufzurufen, eine Flugreservierung vorzunehmen, elektronische Post zu versenden, Schach zu spielen, einen Hurrikan zu simulieren oder die Zusammenstellung des Inventars einer Fabrik aktuell zu halten.

Die Instruktionen, aus denen das Programm besteht, müssen in einer Sprache geschrieben sein, die der Prozessor lesen kann. Wenn man ganz genau sein will, so lesen Prozessoren nur ihre eigene „Maschinensprache". („Lesen" bedeutet in diesem Fall „sind in der Lage, auszuführen".) Der „Instruktionssatz" eines Computers ist eine bestimmte Menge von Ketten binärer Zahlen, die für den Prozessor eine festgelegte „Bedeutung" haben. Wenn sie dem Prozessor als Input zur Verfügung gestellt werden, veranlassen sie die Schaltungen des Prozessors dazu, bestimmte Aktionen auszuführen.[84]

Maschinensprache: Ketten von Einsen und Nullen

Programme in Maschinensprache sind also Programme, die in Ketten von Einsen und Nullen ausgedrückt sind. Jedes maschinensprachliche Programm besteht aus einer Folge von Instruktionen. Die Instruktionen haben allgemein ausgedrückt die Form von

(1) einem Kommando an den Prozessor, das ihn dazu auffordert, eine bestimmte Operation auszuführen,
(2) der Identifikation von Parametern, auf die hin die Operation auszuführen ist, und
(3) der Angabe der Speicherstellen im Prozessor, an denen bestimmte Aktionen stattfinden sollen.[85]

Eine derartige Instruktion könnte aus ihrer binären Form rückübersetzt etwa lauten: „Bewege Parameter A zu der Speicherstelle 1103".

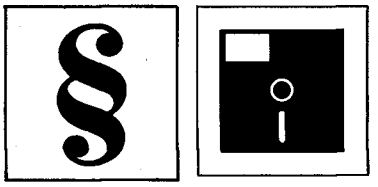
Maschinensprachliche Programme enthalten eine große Anzahl derartiger Instruktionen, die alle als Abfolgen von Einsen und Nullen ausgedrückt sind. In den ganz frühen Tagen der Computer-Industrie, schrieb nahezu jeder, der Computerprogramme schrieb, diese direkt in Maschinensprache, weil dies der einzige Weg war, der Maschine Instruktionen zu erteilen. Es handelte sich dabei um einen langwierigen, mühsamen und fehleranfälligen Prozeß. Wegen dieser Probleme beim Schreiben von Programmen in Maschinensprache, begannen einige damit, Programme zu schreiben, die die Aufgabe des Programme-Schreibens erleichterten, indem sie als Übersetzer zwischen den Symbolen, die Menschen leicht verstehen können, und der Maschinensprache dienten.

„Übersetzungsprogramme"

Diese Übersetzungsprogramme führen Aufgaben aus, die in gewisser Weise den Aufgaben ähnlich sind, die Übersetzer (etwa bei den Vereinten Nationen) ausüben. Die Programmierer schreiben ihre Programme in von der Maschinensprache abweichenden anderen Sprachen, in Sprachen, die für sie natürlicher sind. Die Übersetzungsprogram-

84 M. Weik, a.a.O. (Fn. 61), bei 80, 296, 315 (mit der Definition „Menge von Operatoren aus dem 'Instruktionsrepertoire' oder 'Operationscodes', zu deren Ausführung der Computer in der Lage ist" für „Instruktionssatz").

85 M. Bohl, Information Processing, 192–93 (4. Aufl. 1984). Die folgende Darstellung im Text hinsichtlich der Entwicklung von den Maschinensprachen hin zu natürlicheren Sprachen beruht auf den Seiten 365–368 bei Bohl.



me übersetzen dann die so erstellten Programme in Maschinensprache. Die Übersetzung findet nicht notwendigerweise gleichzeitig statt. Mit Ausnahme der Programme, die man als „Interpreter" bezeichnet, bereiten die Übersetzungsprogramme im allgemeinen vollständige Übersetzungen anderer Programme vor und präsentieren dann die vollständige Übersetzung dem Prozessor für Zwecke der Abarbeitung. Die im Rahmen derartiger Übersetzungsprogramme zur Verfügung stehenden Sprachen können folgendermaßen kategorisiert werden:

Assembler: Eine „verständlichere" Maschinensprache

Bei der Assembler-Sprache handelt es sich um Maschinensprache, die allerdings mit Symbolen geschrieben wird, die für Menschen verständlicher sind als die binäre Notation.[86] Anders ausgedrückt, die Befehle sind dieselben wie in der Maschinensprache, aber sie werden mit Hilfe von Buchstaben oder erkennbaren Worten (wie z.B. „ADD") anstelle von Einsen und Nullen ausgedrückt. Parameter können als Dezimal- oder Hexadezimalzahlen anstelle von Binärzahlen ausgedrückt werden.[87] Speicherstellen erhalten Namen, wobei diese Namen so ausgewählt werden können, daß sie eine Vorstellung von dem vermitteln, was in der Speicherstelle gespeichert wird, also z.B. „Bilanz" oder „Summe" oder „Gesamtzahl". Eine weitere Hilfe für die Lesbarkeit von Assembler-Programmen ergibt sich daraus, daß der Programmierer Kommentare zur Begründung jeder Instruktion beigeben kann.[88] Die Kommentare werden für Zwecke späterer Auswertung durch den Autor oder irgendeinen anderen Leser geschrieben. Während der Ausführung des Programms werden sie vom Computer nicht verarbeitet.[89] Programme in Assembler-Sprache können vom Computer nicht direkt gelesen werden. Ein Programm (als „Assembler" bezeichnet) übersetzt die in Assembler-Sprache geschriebenen Programme in die Einsen und Nullen der Maschinensprache.[90]

Binär	*Hexa-dezimal*	*Dezimal*
0000	*= 0*	*= 0*
0001	*= 1*	*= 1*
0010	*= 2*	*= 2*
0011	*= 3*	*= 3*
0100	*= 4*	*= 4*
0101	*= 5*	*= 5*
0110	*= 6*	*= 6*
0111	*= 7*	*= 7*
1000	*= 8*	*= 8*
1001	*= 9*	*= 9*
1010	*= A*	*= 10*
1011	*= B*	*= 11*
1100	*= C*	*= 12*
1101	*= D*	*= 13*
1110	*= E*	*= 14*
1111	*= F*	*= 15*

Sowohl maschinensprachliche Programme als auch Programme in Assembler-Sprache sind auf der Art und Weise aufgebaut, in der ein Computer konstruiert worden ist, und nicht auf der Art und Weise, wie Menschen denken. Sie spezifizieren jede elementare Aktion, die der Computer ausführen muß, was eine sehr mühsame Methode der Instruierung ist.[91] Obwohl das Programmieren in Assembler-Sprache eine bedeutende Verbesserung im Vergleich zum Schreiben in Maschinensprache darstellt, ist es doch noch mühsam, weil (allgemein ausgedrückt) jede vom Prozessor auszuführende Instruktion vom Programmierer niedergeschrieben werden muß. In den 50er Jahren wurde man darauf aufmerksam, daß man Programme schreiben kann, die einzelne Instruktionen hohen Niveaus in eine Serie von assembler- oder maschinensprachlichen Instruktionen übersetzen können.

„Hochsprachen": Für Menschen lesbare Programme

Auf der genannten Idee aufbauende „Hochsprachen" beruhen auf dem Gedanken, daß die Aufgabe der Übersetzung eines in menschlicher Denkweise geschriebenen Programms (obere Ebene) in ein dem Design des Computers entsprechendes Programm (untere Ebene) mechanisiert werden kann, wenn die Hochsprache ausreichend formalisiert ist.[92]

Die erste formalisierte Hochsprache, die kommerziell verfügbar war, war FORTRAN (Formula-Translator), die bei IBM in den späten 50er Jahren erstellt wurde. Seitdem sind viele andere geschaffen worden, mit Namen wie COBOL, BASIC, PASCAL und LISP.[93]

Genauso wie Programme in Assembler-Sprache durch „Assembler" in die Maschinensprache übersetzt werden, werden Programme in einer Hochsprache durch „Compiler" genannte Programme in die Assembler-oder Maschinen-Sprache übersetzt.[94] Dadurch, daß eine einzelne Instruktion der Hochsprache in mehrere Instruktionen der „unteren Ebene" expandiert wird, nehmen Compiler vom Programmierer die Last, wissen zu

86 A.a.O.

87 Eine Hexadezimalzahl ist eine Zahl, die anders als eine Dezimalzahl nicht auf dem Zehner-, sondern auf dem Sechzehner-System (Basis 16 statt 10) beruht. Jede Gruppe von vier binären Zahlen kann als eine hexadezimale Zahl ausgedrückt werden (a.a.O., bei 77–79). Beispielsweise entspricht hexadezimal 11 im Dezimalsystem 17. Hexadezimal 1A ist dasselbe wie dezimal 26.

88 The MacGraw-Hill Computer Handbook, a.a.O. (Fn. 54), bei § 11.3.

89 S. Alagic/M. Acbib, The Design of Well-Structured and Correct Programs 11, 17, 254 (1978).

90 M. Bohl, a.a.O. (Fn. 85), bei 365.

91 A.a.O., bei 365–368. Vgl. auch T. Pratt, Programming Languages: Design and Implementation 21 (2. Aufl. 1984).

92 M. Bohl, a.a.O. (Fn. 85), bei 370–71.

93 Vgl. History of Programming Languages Conference 25–41 (Hrsg. v. R. Wexelblatt, 1981).

94 T. Pratt, a.a.O. (Fn. 91), bei 21. Vgl. auch M. Bohl, a.a.O. (Fn. 85), bei 372–374.

Der menschliche Leser

müssen, wie ein bestimmter Prozessor tatsächlich arbeitet (d.h., woraus sein „Instruktionssatz" besteht).[95] Als Ergebnis davon kann der Programmierer das Programm in einer Sprache ausdrücken, die einmal wesentlich zeitökonomischer ist und zum anderen für den menschlichen Leser verständlich. Hinzu kommt, daß die Existenz von Compilern beispielsweise für eine populäre Sprache wie COBOL es erlaubt, daß dieselben Programme auf Prozessoren mit sehr unterschiedlichen Instruktionssätzen laufen. Anders ausgedrückt: Für den jeweiligen Prozessor geschriebene Compiler können dasselbe in Hochsprache ausgedrückte Programm in verschiedene Maschinensprachen übertragen.[96]
In Hochsprachen kann das Instruktionsformat „Kommando-Parameter-Speicherstelle" beträchdich liberaler gehandhabt werden. Die direkten Befehle von Sprachen der unteren Ebene (z.B. „Addiere A,B", „Bewege X,Y") werden durch verfeinerte Anweisungen ersetzt.
(Z.B.: „Wenn a größer als 1000 wird,
dann drucke 'fortfahren';
sonst drucke 'stop'".)
In Frage kommen auch Beschreibungen von Relationen zwischen „Objekten" und Suchworten, die es erlauben, daß aus diesen Relationen Folgerungen gezogen werden. Eine weitere Möglichkeit besteht darin, graphische Ikonen und Bildschirmzeiger zu verwenden.[97]

Das zentrale Argument: Programme sind „Lektüre"

Das Argument, um das es hier geht, hat zwei Aspekte.

1. Computer können in Sprachen programmiert werden, deren Verwendung in zunehmendem Maße für Menschen „natürlich" ist. Deswegen beruht jedes Argument, das darauf abzielt, man müsse Programme anders als andere literarische Werke behandeln, weil Programmiersprachen sich von menschlicher Sprache unterscheiden, auf einer vollständig falschen Prämisse.

2. Es gibt keine linguistische Basis für eine Unterscheidung zwischen dem Schutz, den man in Hochsprachen ausgedrückten Programmen zubilligt, und dem Schutz, den man Programmen gewährt, die in Maschinen- oder Assembler-Sprache geschrieben sind. Der Grund liegt darin, daß eine maschinensprachliche oder assemblersprachliche Version eines Hochsprachenprogramms nichts anderes als eine Übersetzung (in vielen Fällen eine mechanische Übersetzung) des Ausdrucks von einer Sprache in eine andere ist.

Anwendungsprogramme vs. Betriebssysteme

c. Die Kategorien der Programm-Werke
Für die Zwecke dieses Beitrags können wir uns zwei große Kategorien von Programmen vorstellen:
Anwendungsprogramme sind Programme, die den Computer dazu instandsetzen, ein bestimmtes Problem eines Kunden zu lösen.[98]
Beispiele dafür sind:
Textverarbeitung,
Flugreisenbuchung,
Berechnung der Einkommensteuer.
Betriebssysteme sind Programme, die den Computer dazu instandsetzen, die anderen Programme laufen zu lassen.[99] Sie fungieren als Schnittstelle zwischen dem Anwendungsprogramm und der Hardware.[100] Sie koordinieren außerdem die vielfältigen Ereignisse, die sich in einem Computersystem abspielen.[101]

95 J. Sammet, a.a.O. (Fn. 41), bei 9.
96 A.a.O., bei 9–10, 36–43.
97 Ein kompakter Vergleich der mit verschiedenen (Hoch-) Programmiersprachen für dasselbe Problem erzielten Lösungen findet sich bei Tesler, Programming Languages, Scientific American, September 1984, bei 70–78.
98 A. Ralston, a.a.O. (Fn. 55), bei 92.
99 A.a.O., bei 1053.
100 Vgl. D. Barron, Computer Operating Systems 5–6 (1971); M. Bohl, a.a.O. (Fn. 85), bei 401–402. Eine kurze aber umfassende Geschichte der Betriebssystem-Entwicklung findet sich dort bei 404–431.
101 Vgl. auch A. Ralston, a.a.O. (Fn. 55), bei 1055–58.