Humanities of the Digital

Philologische Perspektiven auf Source Codes als Beitrag einer computerarchäologischen Knowledge Preservation

Stefan Höltgen

0. Einleitung

Der digitale Wandel hat alle Bereiche des Wissens erfasst. Informationen kursieren als Daten in Netzwerken und stehen an beinahe jedem Ort der Welt auf Computern zur Verfügung. Klassische Medieninhalte haben sich von ihren materiellen Substraten gelöst und werden als Dateien oder Datenstreams auf mobilen Endgeräten oder computerisierten Großbildschirmen wiedergegeben. Die Software selbst, die solche Geräte in Form von Apps oder Programmen zur Konversion des jeweiligen Mediums im Digitalcomputer benötigt, ist als Sourcecode in algorithmisch sorgsam edierten Repositories abgelegt. Papier, so scheint es, wird langsam zu einem medientechnischen Relikt und Schrift residiert häufig nur noch als grafischer Gestalteffekt auf Displays.

Nicht trotz, sondern wegen dieser eskalierenden Digitalisierung und Virtualisierung unserer Symbolwelten wächst das Schrifttum in geometrischem Maße, denn »[d]er Computer ist eine Technologie der symbolischen Repräsentation und der Kommunikation, kurz – eine

Technologie des Schreibens.«[1] Mit ihm wird immer mehr Text geschrieben, weil auf immer mehr digitalen Kanälen produziert und rezipiert werden kann, und *weil diese Kanäle zur Hälfte*[2] *selbst aus Schrift gebaut sind*: aus Software – das sind Programmcodes und Daten, die unsichtbar in die Unterflächen der Digitaltechnologie geschrieben dafür sorgen, dass auf den Oberflächen etwas lesbar wird.[3] Die Bedeutung von Schrift im ›Zeitalter der Digitalisierung‹ kann also kaum unterschätzt werden, treibt sie doch all unsere symbolverarbeitenden Maschinen an und bringt damit *en passant* eine eigene, arkane Schriftkultur hervor, die als solche und jenseits der ökonomischen Verwertbarkeit ihrer Texte bislang kaum Beachtung gefunden hat. Dabei scheint evident, dass in nunmehr 180 Jahren[4] Programmiersprachen- und 80 Jahren Computergeschichte etliche dedizierte Schriftkulturen entstanden sein müssen, die ihre eigenen Sprachen, Formen, Stile und Publikationsarten und -kanäle hervorgebracht haben.

Mein folgender Beitrag will dieser Evidenz nachgehen und Programmiersprachen und Programme als Elaborate einer Textkultur verstehen, die nicht nur mit informatischen, sondern auch mit philologischen Methoden (er)fassbar gemacht werden können und sollten. Am Beispiel der in den 1970er und 1980er Jahren am weitesten verbreiteten

1 David J. Bolter: Das Internet in der Geschichte des Schreibens. In: Stefan Münker und Alexander Roesler (Hg.): *Mythos Internet.* Frankfurt a.M.: Suhrkamp 1997, 37–55, hier: 37.

2 Zur irreduziblen anderen Hälfte vgl.: Friedrich Kittler: Hardware. Das unbekannte Wesen. In: Sibylle Krämer (Hg.): *Medien, Computer, Realität: Wirklichkeitsvorstellungen und neue Medien.* Frankfurt a.M.: Suhrkamp 1998, 119–132.

3 Zum medienästhetischen und -technischen Begriffspaar Unterfläche/Oberfläche vgl. Frieder Nake: Das doppelte Bild. In: Margarete Pratschke (Hg.): *Bildwelten des Wissens. Kunsthistorisches Jahrbuch für Bildkritik.* Band 3, Nummer 3: Digitale Form. Berlin: Akademie-Verlag 2005, 40–50.

4 Setzt man die Ausführungen in Ada Lovelaces »Notes«, in denen sie einen Algorithmus zur Berechnung der Bernoulli-Zahlen für eine künftig zu bauende Rechenmaschine entwirft, als deren Beginn (vgl. Tanja Rahneberg: Ada Lovelace und das erste Computerprogramm der Welt. In: *Max-Planck-Gesellschaft*, https://bit.ly/3Xirkf3 [21.06.2023, 16:00]).

Programmiersprache BASIC soll gezeigt werden, wie sich Erkenntnisse aus der Lektüre von Programmcodes über diese Kultur und ihre Praktiken gewinnen ließen. Nach einer einführenden Betrachtung über die Attribuierung von Programmiersprachen als Sprachen findet eine philologische Annäherung an Sourcecodes auf Basis sprach- und literaturwissenschaftlicher Einordnungen statt. Die Betrachtung steuert auf das Ziel zu, Programmiersprachen, Programmierer:innen und die zugehörigen Computer als kulturelle Artefakte zu markieren, deren Schrifttümer es mit spezifischen Mitteln zu erforschen und bewahren gilt.[5] Anhand von Archivmaterialien sollen dabei die spezifischen philologischen Aspekte von Sourcecodes vorgeführt werden.

1. Die Sprache(n) der Medien

Der häufig anzutreffenden[6] Aussage, Programmiersprachen seien gar keine Sprachen, lässt sich bei genauerem Hinsehen widersprechen. Der Eindruck eines *kategorialen* Unterschiedes entsteht vor allem dadurch, dass 1. Programmiersprachen nicht von Menschen gesprochen und 2. natürliche Sprachen von Maschinen nicht verstanden würden. Das Missverständnis scheint in den Begriffen »sprechen« und »verstehen« zu liegen, die eine homozentristische Perspektive insinuieren, die in beiden Fällen meint, dass Kreativität in der Produktion und Rezeption von Sprache an den engen Grenzen maschineller Sprachverarbeitung scheitern müsse. Allerdings findet derzeit nicht nur eine ›Quantifizierung‹ dieses Verstehensbegriffs statt, bei dem die Emergenz von Bedeutungen als Statistik in neuronalen Netzen emuliert werden kann; Menschen haben auch immer schon mit nicht-natürlichen Sprachen kommuniziert. Nicht nur Maschinen tauschen sich (über technische

5 Das dahinter stehende Vorhaben und sein Ziel werden von mir derzeit im Rahmen eines Forschungsprojektes an der Universität Bonn verfolgt. (http://txt3.de/basic [zuletzt abgerufen am 21.06.2023]).

6 Z. B.: Gero von Randow: Reden Sie mit Ihrem Computer? In: *Zeit Online*, 04.06.2014, https://bit.ly/44pRIGj (03.07.2023, 9:45).

Kanäle und standardisierte Datenprotokolle) seit Anbeginn des Computerzeitalters aus; auch Menschen kommunizieren mit Maschinen formal (programmieren) wie informell (Chatbots) und verständigen sich sogar untereinander mit nicht-natürlichen Sprachen[7] – anders wäre die heute übliche gemeinschaftliche Entwicklung von Software (aber auch von mathematischen Theorien und Beweisen) kaum möglich. Dass es sich bei Programmiersprachen zumeist um geschriebene (oder gezeichnete[8]) Sprachen handelt, ist ebenfalls kein hinreichendes Argument gegen ihre Sprachhaftigkeit, gibt es doch auch natürliche Sprachen, die nicht gesprochen, sondern nur geschrieben oder gestikuliert werden.[9] Und schließlich kann das von v. Randow geäußerte Kriterium, dass sich mit Programmiersprachen keine Gefühle ausdrücken ließen, zumindest angezweifelt werden.[10]

---

7 Im Rahmen meiner Forschung untersuche ich beispielsweise Leserbriefkommunikation in Computerzeitschriften (die heute in Internet-Foren weitergeführt wird) und finde immer wieder Beispiele, in denen die Kommunikate der menschlichen Schreiber:innen/Leser:innen nur noch im Austausch von Code-Fragmenten bestehen, die von den Kommunikationspartner:innen als Propositionen aufgefasst werden.

8 Schaltpläne, Flussdiagramme und diagrammatisch angelegte Programmiersprachen (wie Konrad Zuses »Plankalkül«, vgl. Raul Rojas: *Konrad Zuse's Early Computers. The Quest for the Computer in Germany.* Berlin: Springer 2023, 147–170 [im Druck]) müssen als (teilweise) nicht symbolische Sprachen in diesen Kontext einbezogen werden, selbst wenn sie (noch) nicht direkt auf Maschinen implementiert und von diesen ausgeführt werden können. (Vgl. Stefan Höltgen: *Open History. Archäologie des Retrocomputings.* Berlin: Kadmos 2022, 155f.)

9 Hier können als Beispiele für ausschließlich geschriebene Sprachen die so genannten »toten Sprachen«, die nur noch in schriftlichen Dokumenten belegt sind, dienen – aber auch Systeme, wie die Gebärdensprache oder grafische Sprachen wie Bliss Symbolics (https://www.blissymbolics.org/, [zuletzt abgerufen am 03.07.2023]) und Emoticons. Vgl. https://de.babbel.com/de/magazine/was-sind-programmiersprachen (zuletzt abgerufen am 03.07.2023).

10 G. von Randow selbst hat in einem früheren Beitrag über das (un)mögliche Bewusstsein von Maschinen eine skeptizistische Gegenfrage gestellt: »[W]oher wissen wir so genau, daß Menschen Bewusstsein haben? Vielleicht, weil sie es bekunden?« (Gero von Randow: *Roboter. Unsere nächsten Verwandten.* Reinbek: Rowohlt 1997, 247f.) Ein Programm mit Ausgabe »Ich habe Gefühle.« müsste

Die Existenz einer pragmatischen Ebene von Programmiersprachen impliziert noch weitere hierfür notwendige linguistische Kategorien, die die Bedingung für ihre Zugehörigkeit zum Phänomenkreis *Sprache* bilden: Mit ihnen müssen sich *kohärente* Formen konstruieren lassen, die sich über *syntaktische* Strukturen (Regeln) zu komplexeren, *semantisch* zusammengehörigen Gebilden kombinieren lassen. Sie müssen *produktiv*[11] sein, so dass sich beliebig viele solcher Sprachgebilde formulieren und variieren lassen. Mit ansteigender Komplexität der sprachlichen Elaborate (Zeichen-Wort-Satz-Text-Genre-Gattung) können *stilistisch* komplexere Varianten derselben Grundaussage und schließlich sogar *Soziolekte*[12] entstehen, die vom Wissen, dem kulturellen Kontext und der historischen Einordnung der Sprachnutzer:innen abhängen.

2. Computerphilologie

Im Rahmen eines Forschungsprojektes zur Archäologie der frühen Mikrocomputer und ihrer Programmierung[13] habe ich u.a. einen Ansatz zur sprach- und literaturwissenschaftlichen Analyse von Programmiersprachen entwickelt. Dazu habe ich Argumente für die Vergleichbarkeit und Abhängigkeit von unterschiedlichen Programmen, die ähnliche

demzufolge zumindest Skepsis über die mögliche Gefühlsfähigkeit des Computers, auf dem es läuft, auslösen.

11 Das Kriterium der Produktivität meint zunächst die Herausbildung neuer Begriffe auf der morphologischen Ebene. Auch dies wir häufig als Argument gegen die Sprachhaftigkeit von Programmiersprachen herangezogen (vgl. Umberto Eco: *In Search for the Perfect Language.* Oxford/Cambridge: Blackwell 1995, 311), obgleich konkatenative Programmiersprachen wie Forth oder Logo gerade darin bestehen, neue »Wörter« (https://www.forth.com/starting-forth/11-forth-compiler-defining-words/, [zuletzt abgerufen am 03.07.2023]) mit komplexeren Bedeutungen herauszubilden.

12 Vgl. F. Naz/J. E. Rice: Sociolinguistics and programming. In: *2015 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. Victoria 2015, 74–79. Hier werden mithilfe von KI-Verfahren stilistische Elemente in Sourcecodes gesucht, die sich als Gendermarkierungen verstehen lassen.

13 Vgl. Höltgen: *Open History*, 129–169.

audiovisuelle Ausgaben erzeugt haben, gesammelt. Die bloße Ähnlichkeit der Oberflächen korrespondiert nicht notwendig auch mit einer Ähnlichkeit ihrer Unterflächen. Im Gegenteil: Auf der Code-Ebene zeigen sich bisweilen kategorische Differenzen in der Art und Weise der verwendeten Programmiersprachen (BASIC, C, verschiedene Assemblersprachen) und Programmiersysteme (Digitalcomputerprogramme, Analogrechnerprogramme), die sich allerdings dennoch auf Verwandtschaften (im Sinne einer *Rezeptionsästhetik*) untersuchen ließen. Hierzu musste ich jedoch zunächst die grundlegenden linguistischen Eigenschaften, die einen solchen Vergleich ermöglichen, herausarbeiten. Dies geschah durch die Analyse von Kohäsionsverfahren[14] und in der Suche nach Merkmalen produktiver Rezeption (hier: von verwendeten Algorithmen)[15].

Aus den Funden und Ergebnissen ergaben sich zahlreiche Anschlussfragen, die ich als Forschungsdesiderat derzeit in einem Projekt über die Programmiersprache BASIC und ihre Programmierkulturen untersuche. Bevor ich diese Fragen und die aus den Philologien eingesetzte Methodologie vorstelle, sollen die Eigenschaften der Programmiersprache BASIC und ihrer Programmierung, die diese Sprache besonders attraktiv für eine solche Untersuchung machen, kurz skizziert werden.

2.1 BASIC

BASIC (Beginner's All-purpose Symbolic Instruction Code) entstand 1964 in den USA als Programmiersprache für Studierende der Geisteswissenschaften. In BASIC wird weitgehend von mathematischen und informatischen Programmiersprachen-Konzepten und -strukturen abstrahiert, so dass sich die Lernenden auf die Entwicklung von Algorithmen und deren Übertragung in Programme konzentrieren konnten. Die Entwickler von BASIC stellten ihre Sprache der Allgemeinheit zur

14 Höltgen: *Open History*, 147–155.

15 Höltgen: *Open History*, 132–138.

Verfügung, was nicht nur dazu führte, dass binnen weniger Jahre zahlreiche Schulen und Universitäten BASIC-Kurse anboten,[16] sondern auch dazu, dass Computerfirmen ab Ende der 1960er Jahre damit begannen, eigene, speziell an ihre Systeme angepasste BASIC-Dialekte zu entwickeln.

Die Dialektvielfalt ›explodierte‹ förmlich ab Mitte der 1970er Jahre, als Mikrocomputer für Privatleute angeboten wurden. Jeder Hersteller verbaute in sein System einen speziell an dessen Hardware angepassten BASIC-Dialekt. Zusätzlich entstanden alternative Dialekte und Spracherweiterungen, mit denen man die jeweiligen Systeme nachträglich ausstatten konnte, um sie für spezifische Anwendungen zu rüsten.[17] Auf diese Weise entstanden bis Anfang der 1990er Jahre[18] Hunderte, wenn nicht Tausende unterschiedliche BASIC-Dialekte, für die sich jeweils eigene User-Gemeinschaften herausbildeten. Programmierer:innen dieser Dialekte tauschten Kenntnisse und Programmcodes untereinander aus, für sie und ihre Plattformen wurde Sekundärliteratur (Zeitschriften und Bücher mit Programmcodes und anderen Darstellungen) publi-

16 Die führte zur oben genannten Weitverbreitung der Sprache: »BASIC was once the lingua franca for commurucatmg with computers.« (Marc Jones Lorenzo: *GOSUB without RETURN. Between the Lines of the BASIC Programing Language.* Philadelphia/Pittsburg: SE Books 2022, 235).

17 Die Vielfalt zeigt die (unvollständige) Liste an BASIC-Dialekten in der Wikipedia (https://de.wikipedia.org/wiki/Liste_von_BASIC-Dialekten [zuletzt abgerufen am 29.06.2023]). Unter den Überschriften zu den Plattformen finden sich (einige) für diese Plattform verfügbaren BASIC-Dialekte.

18 Mit dem Erscheinen der letzten Homecomputer (die sich durch speziell für den Anschluss an heimische Medientechnologie angepasste Schnittstellen und eine Hardware- und Software-Vollausstattung auszeichneten) Anfang der 1990er Jahre endete zwar nicht die Verwendung der Programmiersprache; mit dem Verlust an Plattformen mit fest eingebauten BASIC-Dialekten diversifizierte sich der Gebrauch der Sprache jedoch so stark, dass Eingrenzungen, wie die genannten, kaum noch möglich waren. Im selben Zeitraum wurden auch Prinzipien der strukturierten Programmierung in neue BASIC-Dialekte eingeführt, die den idiosynkratischen Gebrauch der Sprache stark (zugunsten einer »informatischeren« Programmierlehre und -praxis) einschränkten, was einen weiteren Verlust an Vielfalt zur Folge hatte.

ziert, es wurden Clubs gegründet und Newsletter/Fanzines herausgegeben und Hardware- und Software-Erweiterungen für das jeweilige System entwickelt.

2.2 Quellenlage

Aus dieser Vielfalt an BASIC-Dialekten ist ein Archiv erwachsen, das als das größte und vielfältigste innerhalb der Computergeschichte gelten dürfte.[19] Die im Zeitraum von 1975 bis 1995 noch weitgehend fehlende oder erst rudimentär verfügbare Vernetzbarkeit[20] von Computern führte dazu, dass Source Codes vorrangig auf Papier publiziert und proliferiert wurden. Neben den bereits erwähnten Zeitschriften und Büchern entstand ein großer Fundus an *grauer Literatur*, in dem ebenfalls Programmcodes in kleineren und größeren Regionen unter den Computer-Nutzer:innen kursierten. Darüber hinaus existiert (in privaten Archiven und Nachlässen) ein prinzipiell nicht zu überschauender und systematisch schwer erfassbarer Fundus an *Ephemera* (Programmausdrucke, handschriftliche Notizen) und *marginalisierten* Schriftstücken, in denen sich individuelle Zugänge zur Programmiersprache und ihrer Programmierung in besonderem Maße zeigen. Hinzu kommen Datenträger (vor allem Disketten und Kassetten), die Programmcodes enthalten.

Die aus dem Ausmaß und der teilweise prekären Quellenlage dieses Archivs erwachsenden Schwierigkeiten für die Bewahrung und Erforschung dieser Textbestände werden noch durch die Tatsache verschärft,

19 Michel J. Halvorson beschreibt diese Amateur-Programmierkultur im Kapitel »Four Million BASIC Programmers«, in: Ders.: *Code Nation: Personal Computing and the Learn to Program Movement in America*. Kentfield: ACM Books/Morgan & Claypool 2020, 127–165.

20 Vernetzbar waren Computer in diesem Zeitraum entweder physikalisch durch den Aufbau von Kabelnetzwerken (was für private Zwecke eher unüblich war) oder via Telefonnetz mit Modems oder Akustikkopplern, was für Privatanwender einerseits kostenintensiv war und andererseits Einschränkungen im Datenumfang und der Übertragungsgeschwindigkeit mit sich brachte, so dass ein Austausch von Wissen, Programmen und Daten, wie er heute üblich ist, auf diesem Weg kaum stattfand.

dass die Quellen in unterschiedlichsten Erhaltungsgraden vorliegen und die auf Papier gespeicherten Informationen zudem unterschiedliche Druckqualitäten aufweisen. Und als genüge dies noch nicht, so bekommt es die Forschung zudem mit Zeichensätzen und Typografien zu tun, die spezifisch für die jeweilige Computerplattform sind[21] und eine Digitalisierung mit Texterkennung (OCR) verkomplizieren. Hier kann zunächst allein ein menschliches Close-Reading der Source Codes erfolgen, das – ausgestattet mit Hintergrundwissen über die jeweilige Plattform und den verwendeten BASIC-Dialekt – Zugänge zur Funktionsweise, zum Aufbau und dann zum Programmierstil aus den Sourcecodes kondensiert. Welche philologischen Kategorien hierbei Berücksichtigung finden, um die anvisierten Ziele des Projektes zu erreichen, soll nun dargelegt und mit Beispielen aus meinem Archiv illustriert werden.[22]

---

21 Anfang der 1960er Jahre einigte sich ein internationales Konsortium auf die Standardisierung von Computerzeichensätzen, womit die Vernetzung unterschiedlicher Plattformen vereinfacht werden sollte. Das ASCII-System (American Standard Code for Information Interchange) beschreibt die ersten 128 Zeichen (7 Bit), die einige Steuerzeichen, das groß- und kleingeschriebene Alphabet, die Ziffern und Satzzeichen umfassen. Homecomputer nutzen diese ASCII-Zeichenkodierung – aber zusätzlich auch noch das 8. Bit, das es ermöglicht weitere 128 Zeichen zu diesem Standardzeichensatz hinzuzufügen. In diesen werden systemspezifische Zeichen, grafische Elemente, internationale Symbole und Steuercodes hinterlegt, die von Plattform zu Plattform unterschiedlich ausfallen und oft nur (wenn überhaupt) von systemeigenen Druckern ausgedruckt werden können.

22 Da die Fragestellung einerseits synchrone Aspekte der Programmiersprache/Programme umfasst (Paratextualität, Individualstil, Textgestaltung, Soziolekt etc.), andererseits genealogische (diachrone) Perspektiven auf Programmiersprache/Programme beleuchtet (Dialektologie, Rezeption, Stemmatologie, Intertextualität etc.) und schließlich transzendentale Elemente des Sprachdesigns fokussiert, werden innerhalb meines Forschungsprojektes bei jeder Kategorie alle drei Perspektiven berücksichtigt. Hier kann dies nur in Ansätzen geschehen.

2.3 Sprachwissenschaftliche Perspektiven

Von den in der Sprachwissenschaft untersuchten Kategorien *Laut, Wort, Satz, Text* sowie *semantische* und *kommunikative* Aspekte entfallen bei der Untersuchung von Sourcecodes die ersten beiden, denn Programmiersprachen werden nicht gesprochen, und bei der Entwicklung eines Programms spielen die morphologischen Eigenschaften der Programmierbefehle keine Rolle.[23] In Hinblick auf k*lassische Grammatik* lässt sich zunächst zeigen, dass in BASIC alle Befehle als englische Verben im Imperativ vorliegen.[24] Interessanter zeigt sich die Konstruktion von Programmzeilen,[25] weil hierbei bereits *syntaktische Strukturen* zum Tragen kommen, die später dafür sorgen, dass die Programmzeile vom BASIC-Interpreter übersetzt werden kann.[26] Aber nicht nur die Lesbarkeit des Codes durch die Maschine wird durch diese syntaktischen

23 Dies ändert sich beim Sprach-Design, wo neben der syntaktischen die morphologische Ebene grundlegend ist.

24 Dies gilt auch für Variablenzuweisungen wie X=X+1, denn hier wäre eigentlich ein (oft fakultativ zu schreibendes) LET davor zu setzen, welches der Zuweisung überhaupt erst ihren Sinn verleiht, indem es den Gleichheitsoperator (=) zu einem Zuweisungsoperator (:=) macht. (Vgl. John G. Kemeny/Thomas E. Kurtz: *Back to BASIC. The History, Corruption, and Future of the Language.* Reading u.a.: Addison-Wesley 1985, 59f.)

25 Diese setze ich hier als Analogon zu den *Sätzen* natürlicher Sprache, betone aber, dass eine Übertragbarkeit der genannten Kategorien von natürlichsprachlichen auf Programmiertexte weder 1:1 möglich ist, noch dem Wesen von Source Codes gerecht würde.

26 Die BASIC-Dialekte der Homecomputer liegen als Interpreter vor: Hierbei wird der Sourcecode erst während der Ausführung (zur Laufzeit) in Maschinencode übersetzt, welchen der Computer dann ausführen kann. Bei Compilern wird zunächst der komplette Sourcecode in Maschinensprache übersetzt und als ausführbares Programm abgespeichert. Diese Verfahren der (maschinellen) Übersetzung bergen weitere interessante philologische Aspekte, die teilweise von der Informatik erforscht werden. Das Fachgebiet der Theoretischen Informatik untersucht in seiner Automaten-Theorie die Frage, wie Berechenbarkeit und formalsprachliche Beschreibung (also Grundlage von konkreten Programmiersprachen) in Zusammenhang stehen. Vgl. Robert Floyd/Richard Beigel: *Die Sprache der Maschinen.* Bonn u.a.: Thomson 1996.

Vorgaben ermöglicht, sondern auch die derjenigen Menschen, die diesen Code schreiben und lesen. Empirische Studien des *Code Reading*[27] haben hier vor allem die Rolle der Kohäsionsverfahren hervorgehoben, die syntaktische Strukturen und semantische Zusammenhänge zwischen einzelnen Programm-Teilen erzeugen.

Kohäsion (und die daraus resultierende *Kohärenz*) sind bereits textlinguistische Phänomene, die Beziehungsstrukturen über den gesamten Programmtext hinweg beschreiben.[28] Ihre Untersuchung ermöglicht den Nachvollzug des Programmaufbaus und liefert Anhaltspunkte über seine ›Klarheit‹ im Sinne des Programmierstils.[29] Die Kohärenz eines Sourcecodes konstituiert das symbolische Gebilde schließlich als einen zusammenhängenden Text, der sich von anderen Texten unterscheiden lässt und sich als Ganzes *literaturwissenschaftlich* in größere Kategorien (Genres, Gattungen etc.) einordnen lässt.

Intertextualität (in ihrem sprachwissenschaftlichen Verständnis) »bezeichnet als theoretischer Begriff […] das Phänomen einer wie auch immer festzulegenden Relation zwischen [unterschiedlichen, S. H.] Texten.«[30] Diese Relationen sind unterscheidbar in Hinblick auf ihren *Grad* (von der Allusion bis hin zum wörtlichen Zitat) und der räumlichen und zeitlichen *Distanz* zum referenzierten Fremdtext. Die linguistische Intertextualitätsforschung liefert für eine Analyse Maße für die Vagheit/Konkretheit und die Enge/Weite der Referenzbeziehungen. Bei BASIC-Sourcecodes lassen sich vielfältige Formen von Intertextualitätsbeziehungen finden und kategorisieren. Dies beginnt bereits damit, dass ein

27 Vgl. Teresa Busjahn/et al.: Eye movement in code reading: relaxing the linear order. In: *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension*, 255–265, sowie Höltgen: *Open History*, 170–172.

28 Vgl. Robert-Alain de Beaugrande/Wolfgang Ulrich Dressler: *Einführung in die Textlinguistik*. Tübingen: Niemeyer 1981, 50–117.

29 Vgl. Paul Nagin/Henry F. Ledgard: *BASIC with Style. Programming Proverbs. Principles of Good Programming with Numerous Examples to Improve Programming Style and Proviciency.* Rochelle Park: Hayden.

30 Susanne Holthius: *Intertextualität. Aspekte einer rezeptionsorientierten Konzeption*. Tübingen: Niemeier 1993, 29.

BASIC-Befehl die hochsprachliche Referenz auf ein Maschinensprache-programm ist, welches ausgeführt wird, wenn der BASIC-Befehl eingegeben worden ist. Das Einbinden[31] von weiteren Programmteilen/Daten in das eigene Programm (etwa durch das Nachladen eines Datenträgers) stellt eine weitere räumlich enge Referenz dar. Für eine Untersuchung der BASIC-Programmierkulturen sind jedoch vor allem räumlich und zeitlich weiter reichende Referenzen interessant. Diese zeigen sich etwa in der *Übernahme* oder *Adaption* fremder Code-Bestandteile in das eigene Programm. Die Existenz von Standardalgorithmen[32] insinuiert bereits, dass es sich hierbei nicht notwendigerweise um eine plagiatorische Praxis handelt: Bestimmte, häufig genutzte Algorithmen (etwa zum Sortieren von Daten oder zum Finden von Informationen in einem Datenbestand) sind Gegenstand von Lehrwerken[33] und dürfen in eigenen Programmen verwendet werden.

Schließlich können für Programmiersprachen auch Aspekte von *Pragmatik* untersucht werden. Hier sind weniger die Sprachhandlungen (die Programme gegenüber Computern in jeden Fall darstellen: als direktive Akte) von Interesse als die Untersuchung von Implikaturen, Präsuppositionen und Methoden der Gesprächsanalyse, mit denen geklärt werden könnte, welche Rolle *Programmiersprachen in der zwischenmenschlichen Kommunikation* spielen und wie sie kommuniziert werden. Hierfür existieren interessante Beispiele, die zeigen, wie BASIC-Programmierer:innen über und mit Code kommunizieren – etwa in Zeitschriften in den Leserbrief-Rubriken. Über solche Code-Konversationen baut sich ein kulturelles Netzwerk auf, dessen sprachliches Medium BASIC und seine Dialekte sind.

---

31 Ein markanter Begriff der Buchherstellung, der hier als *terminus technicus* in die Praktische Informatik eingewandert ist.

32 Ein Begriff aus der Mathematikdidaktik, der standardisierte Rechenwege beschreibt. In der Informatik gibt es Bibliotheken mit Standardalgorithmen für ganz unterschiedliche Probleme, die in unterschiedlichen Sprachen z.B. hier gesammelt werden: https://bit.ly/3]FmcMB (zuletzt abgerufen am 03.07.2023).

33 Für BASIC ist hier etwa das zweibändige Werk von F. R. Ruckdeschel: *BASIC Scientific Routines*, Vol. 1 & 2. Peterborough: Byte/McGraw-Hill 1981 einschlägig.

2.4 Literaturwissenschaftliche Perspektiven

Intertextualität wird auch innerhalb der Literaturwissenschaften untersucht – hier allerdings weniger formal als in der Textlinguistik. Vielmehr hat die poststrukturalistische und postmoderne Literaturtheorie das Vorliegen von Intertextualität als Argument für die grundsätzliche Vernetztheit von Sprache[34] herangezogen. Den diesbezüglich eher sprachphilosophisch gehaltenen Ausführungen der Diskursbegründer:in Julia Kristeva[35] und Jacques Derrida[36] stellt ab Mitte der 1980er Jahre der französische Literaturhistoriker Gerard Genette eine detaillierte und strukturierte Phänomenologie literarischer Text-Text-Beziehungen, die er als *Paratextualität*[37] bezeichnet und von welcher Intertextualität[38] einen Sonderfall bildet, gegenüber. Die Beziehungen, die ein Sourcecode zu anderen Texten haben kann, sind auch hier vielfältig. So finden sich *Epitexte* (die einleitenden Sätze, die Überschriften, Kommentare der in Zeitschriften abgedruckte Sourcecodes usw., vgl. Abb. 1) und *Peritexte* (nachfolgende Leserbriefe, Errata, spätere Code-Versionen usw.). Auch solche paratextuellen Beziehungen von Sourcecodes liefern Argumente für eine kulturelle Auseinandersetzung mit Programmiersprachen und Computerprogrammen.

34 Petra Steiner: *Intertextualität*. In: Stefan Höltgen/Patrick Baum (Hg.): *Lexikon der Postmoderne. Von Abjekt bis Zizek.* Bochum: Projektverlag 2010, 99–101.

35 Julia Kristeva: Wort, Dialog und Roman bei Bachtin (1967). In: Jens Ihwe (Hg.): *Literaturwissenschaft und Linguistik. Ergebnisse und Perspektiven.* Band 3: Zur linguistischen Basis der Literaturwissenschaft II. Frankfurt a.M.: Athenäum 1972, 345–375.

36 Jacques Derrida: Die Différance. In: Ders.: *Randgänge der Philosophie.* Hg. v. Peter Engelmann. Passagen: Wien 1988, 29–52.

37 Gerard Genette: *Paratexte. Das Buch vom Beiwerk des Buches.* Frankfurt a.M.: Suhrkamp 2001.

38 Gerard Genette: *Palimpseste. Die Literatur auf zweiter Stufe.* Frankfurt a.M.: Suhrkamp 1993, 10f.

Abb. 1: Sourcecode mit Paratexten in einer Computerzeitschrift.[39]

Hinterhalt

Eine Spielfigur geht auf Reisen. Sie will möglichst weit herumkommen, muß aber Hinterhalte vermeiden. Das Programm läuft auf einem Spectrum mit 16-KByte-Speicher.

Die Bewegung der Spielfigur übernehmen die Richtungstasten "5" bis "8". Bei jedem Schritt erscheint vor, hinter oder neben ihr ein Hinterhalt (rotes Feld), der nicht betreten werden darf. Ist die Spielfigur völlig eingekreist, ist die Reise bald zu Ende, denn ein Hinterhalt ist unausweichlich. Sie kann nur noch ihre nähere Umgebung abgehen. Ein Schrittzähler informiert zum Schluß über die zurückgelegte Strecke.

Jochen Hartig

```
  5 REM  * Jochen Hartig * 1983
  6 REM  ■■■■■■■■■■■■■■■■■■■■
 10 RANDOMIZE
 15 BORDER 2: CLS
 20 LET y=INT (RND*12)+5
 30 LET x=INT (RND*22)+5
 40 PRINT INK 2;■■■■■■■■■■■■■■■■■■■■
 50 FOR q=1 TO 20: PRINT INK 2;
AT q,0;"■";AT q,31;"■": NEXT q
 70 PRINT INK 2;■■■■■■■■■■■■■■■■■■■■
 80 LET p=-1
150 PRINT AT y,x;"*"
160 LET p=p+1
```

```
170 IF INKEY$<>"" THEN GO TO 17
0
180 LET a$=INKEY$
190 IF a$<"5" OR a$>"8" THEN GO
TO 180
200 PRINT AT y,x;" "
210 IF a$="5" THEN LET x=x-1
220 IF a$="6" THEN LET y=y+1
230 IF a$="7" THEN LET y=y-1
240 IF a$="8" THEN LET x=x+1
250 IF ATTR (y,x)=58 THEN GO TO
400
270 PRINT AT y,x;"*"
300 LET c=INT (RND*4)+1
310 LET x1=x+(c=1)-(c=2): LET y
1=y+(c=3)-(c=4)
320 IF ATTR (y1,x1)=58 THEN GO
TO 300
330 PRINT AT y1,x1; INK 2;"■"
350 GO TO 150
400 PRINT AT 0,1; PAPER 2; INK
7;p
405 FOR q=60 TO -25 STEP -1
410 BEEP .002,q
420 NEXT q
450 INPUT a$
460 RUN
```

Nr. 12 – Dezember 1983 HC 43

Eine Weiterung der linguistischen Betrachtung stellt die Einordnung des Textes als *Textsorte* dar und kaum etwas spricht dagegen, Sourcecodes als dezidierte Textsorte zu behandeln: Sowohl textinterne Kriterien (Lexik, Syntax, Textaufbau und grafische Anmutung) als auch textexterne Kriterien (Funktion, Umfang, Medium, das den Text enthält, situative Einbettung) sind weitgehend spezifisch für Programmcodes.[40] Daher wäre in der Folge zu fragen, ob sich neben der texttypologischen auch weiterreichende *literaturwissenschaftliche Klassifikationen* zur Einordnung anbieten – etwa ob es *Gattungs-* oder *Genre*-Einteilungen geben

39 Jochen Hartwig: Hinterhalt. In: HC – Mein Home-Computer (12/1983), 43. Hier zeigen sich auch die typische Druckqualität und die Verwendung nichtstandardisierter Sonderzeichen.

40 Daniel Siebrecht: Textsorten im Informatikunterricht. In: https://ddi.uni-wuppertal.de/website/repoLinks/v50_vortrag.pdf (zuletzt abgerufen am 29.06.2023), 15f. Die Typologie von Textsorten, die zumeist nach ihrer Funktion geordnet werden, bleibt für Programmtexte allerdings ebenso unscharf wie für natürliche Sprachen (vgl. de Beaugrande/Dressler 1981: *Textlinguistik*, 189–193), denn die Ausführung eines Programmtextes durch einen Computer ist zwar seine »dominante Funktion« (ebd., 191), aber nicht die einzige.

könnte, die im Zusammenhang oder im Kontrast zu Oberflächen-Beschreibung (»Computerspiel«, »Adventure«) stehende Kategorien anbieten. Hier wären für BASIC-Programme etwa der Zeilenumfang des Codes[41] (vgl. Abb. 2), die Einbettung von maschinennahen Operationen im Code[42] oder die Strukturiertheit des Programms[43] Kriterien für eine Zuordnung.

Abb. 2: Ein »BASIC-Oneliner«-Programmcode.[44]

```
1 BORDER 0:POKE 23693,1:CLS :DIM P(10):FOR L=0 TO 4:RESTORE :REA
D W,S,M$,M$:FOR T=1 TO 10:LET P(T)=85-9*T:PRINT AT T,0;M$(T*W-11
 TO T*W):NEXT T:LET M$(P(1)+1)="":LET K=113-CODE INKEY$:FOR A=1
TO L+2:LET G=P(A):LET I=4*RND-2:LET I=(A=1)*((K=1)-(K=2)+W*((K=1
6)-(K=0)))+SGN I*(1+11*(I*I(1))*(G(>P(1)):LET C=CODE M$(G+I+1)-4
5:LET P(A)=G+I*(C(2)-I*(M$(G-I+1)("0")*(C)1)*(A)1):PRINT AT G/W+
.5,G-W*INT (G/W);M$(G+1);AT W,0;L*58+S;AT P(A)/W+.5,P(A)-W*INT (
P(A)/W); INK 7-A*(T(1);"@@@@@@"(A):LET T=T-1+(C=-9*A)*(37-T):LET
 S=S+(A=C):IF A=1 OR P(A)-P(1) OR T>0 THEN NEXT A:POKE (S(56)*23
620,W:NEXT L: DATA 12,1,"OOOOOOOOOOOO...O......OO.O..OO.OO.OO.O
O...$.O.OO....OO....",M$+M$(W TO )+M$
```

41 Hier existieren zum Beispiel etwa »Zwanzigzeiler« oder »One-Liner« als Programmierherausforderungen.

42 Bestimmte BASIC-Befehle erlauben Hardware-Zugriffe. Diese reichen von Manipulationen einzelner Speicheradressen bis hin zur Implementierung von Maschinensprache-Programme mithilfe von BASIC-Laderoutinen.

43 Die mögliche Unstrukturiertheit von BASIC wurde von der Informatikdidaktik früh kritisiert (vgl. Arbeitskreis Schulsprache: Empfehlung, Ergebnisse, Abschlussbericht und Stellungnahme zu BASIC. Paderborn: FEoLL 1976, C1-C14). Spätere Lehrbücher zur strukturierten BASIC-Programmierung hatten das Ziel, leichter les- und wartbaren Sourcecode zu ermöglichen (zum Beispiel: Johann Weilharter: Spaß mit Algorithmen. Einführung in das strukturierte Programmieren mit 42 BASIC-Programmen. Wiesbaden: Sprinter 1984).

44 Dr. BEEP: *Tiny Pacman*. In: *RETRO – BASIC-Sonderheft* (2013), 15; ebenso in: https://rtro.de/tinypacman (zuletzt abgerufen am 03.07.2023). Dieses Programm implementiert mit einer einzigen BASIC-Programmzeile eine Variante des Spiels »Pac-Man« auf dem ZX-Spectrum-Computer.

Zwei zentrale Punkte der Erforschung von Programmierkulturen liegen in der *Editionsphilologie* einzelner Programmcodes und in der *Rezeptionsgeschichte/Stemmatologie* spezifischer Code-Elemente und -Genres. Die editionsphilologische Perspektive nimmt die Genealogie eines einzelnen Programmcodes in den Blick und versucht über das *Versioning* aus den Codefragmenten Aspekte informatischer Selbstausbildung und algorithmischen Denkens der Programmierer:innen herauszulesen. Ein Computerprogramm, zumal wenn sein Sourcecode umfangreich ist, wird nicht linear und selten an einem Tag entwickelt. Zur Sicherung werden die Zwischenstände (manchmal unter Angabe der Versionsnummer oder des Datums im Dateinamen) abgespeichert, um sie später wieder laden und weiterentwickeln zu können. Vereinzelt existieren von Programmen noch die Entwicklungsversionen – gespeichert auf Datenträgern oder als Ausdrucke, die im Idealfall handschriftliche Korrekturen und Annotationen des/der Programmierer:in enthalten. Der kritische Vergleich solcher Versionen mit dem finalen Ergebnis kann wertvolle Informationen über die Entstehungsgeschichte des Programmtextes liefern. Dort, wo diese Sicherungen nicht existieren, geben die Sourcecodes manchmal Hinweise auf ihre Bearbeitungsstufen. (Vgl. Abb. 1, wo die BASIC-Zeilen 15 und 405 offenbar nachträglich eingefügt wurden, während die Zeilen 60, 80, 260 und ggf. weitere nachträglich gelöscht worden sein könnten.)

Wie bereits beim Thema Intertextualität angesprochen, finden nicht selten Übernahmen von fremden Codefragmenten (oder auch noch nicht kodierten Standardalgorithmen) in das eigene Programm statt. Die *historische Rezeptionsästhetik* – und dort vor allem die Ausführungen zur »produktiven Rezeption«[45] – liefern Ansätze, die es ermöglichen, Autor:innen-Leser:innen-Beziehungen zu rekonstruieren. Aus konkreten Übernahmen und mutmaßlichen Ähnlichkeiten

45 Gunter E. Grimm: *Rezeptionsgeschichte. Grundlegung einer Theorie.* München: Fink 1977, 147–153. Wenngleich auch hier der Vorwurf von Vagheit bei der Zuschreibung im Raum steht. (Vgl. Maria Moog-Grünewald: Einfluss- und Rezeptionsforschung. In: Manfred Schmeling (Hg.): *Vergleichende Literaturwissenschaft. Theorie und Praxis.* Wiesbaden: Athenaion 1981, 49–72, besonders: 58–65.)

lassen sich diachrone Vernetzungen in Programmierkulturen ablesen und aus Abweichungen und Änderungen übernommener Vorlagen die epistemologische und ästhetische Progression spezifischer Programme und ihrer Codes ablesen. Wie oft kritisiert[46], sind die Analysekriterien hierfür nur vage formuliert und belegbare produktive Rezeption eher die Ausnahme. Dennoch kann der Blick aus dieser Perspektive interessante Hinweise für die Eingrenzung von Coding Communities (in denen dieselben »Urtexte« kursieren) erbringen. Das aus der diachronen Editionsphilologie stammende Programm der *Stemmatologie* sucht dabei nach Anhaltspunkten für die wahrscheinlichsten Überlieferungsabhängigkeiten. Hier kommen bereits computerisierte Verfahren der Identifikation von Übernahmen zur Anwendung,[47] die sich (etwa im Rahmen eines Digital-Humanities-Forschungsprogramms) auch für Sourcecodes anbieten.

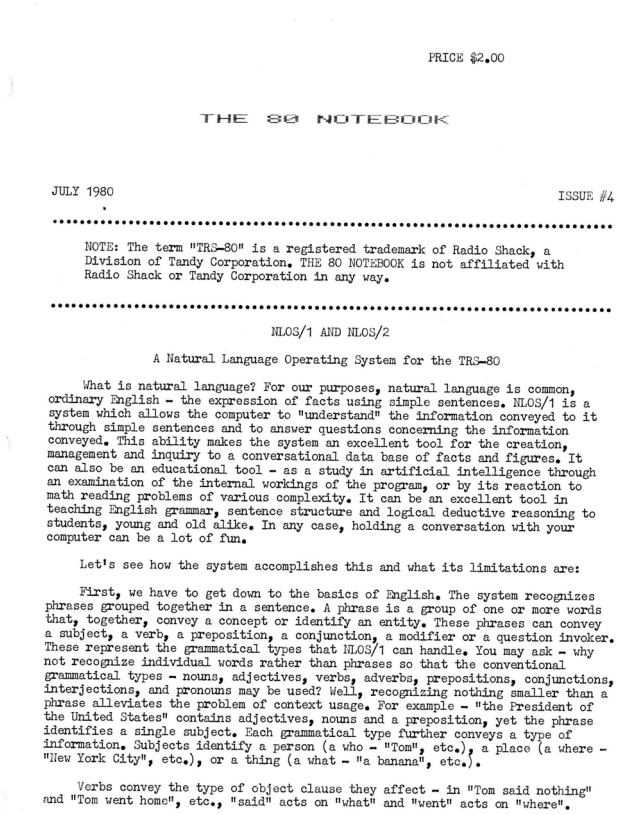
2.5 Bibliothekswissenschaftliche Perspektiven

Schließlich müssen hier noch einige Überlegungen aufgeführt werden, die sich mit der Materialität und Nutzung von Quellen befassen. Da ein Großteil der BASIC-Sourcecodes auf Papier vorliegt, erscheint eine Kategorisierung dieser Quellen sinnvoll. Sie kann einerseits Hinweise auf die Diskursivität von Sourcecodes geben (diese reicht von internationaler Verbreitung bis hin zu alleiniger Nutzung durch den/die Autor:in). Andererseits hilft eine solche Einordnung auch bei der epistemologischen Analyse der Texte oder Textfragmente.

46 z.B. Maria Moog-Grünewald: Einfluss- und Rezeptionsforschung. In: Manfred Schmeling (Hg.): *Vergleichende Literaturwissenschaft. Theorie und Praxis.* Wiesbaden: Akad. Verl. Athenaion 1981, 49–72, hier: 54f.

47 Armin Hoenen/Gerrit Brüning: Überlegungen zur Stemmatologie neuerer Überlieferungen. In: https://t.ly/gOch (zuletzt abgerufen am 29.06.2023).

Abb. 3: Beispiel eines privat produzierten Club-Magazins.[48]



PRICE $2.00

THE 80 NOTEBOOK

JULY 1980 ISSUE #4

NOTE: The term "TRS-80" is a registered trademark of Radio Shack, a Division of Tandy Corporation. THE 80 NOTEBOOK is not affiliated with Radio Shack or Tandy Corporation in any way.

NLOS/1 AND NLOS/2

A Natural Language Operating System for the TRS-80

What is natural language? For our purposes, natural language is common, ordinary English - the expression of facts using simple sentences. NLOS/1 is a system which allows the computer to "understand" the information conveyed to it through simple sentences and to answer questions concerning the information conveyed. This ability makes the system an excellent tool for the creation, management and inquiry to a conversational data base of facts and figures. It can also be an educational tool - as a study in artificial intelligence through an examination of the internal workings of the program, or by its reaction to math reading problems of various complexity. It can be an excellent tool in teaching English grammar, sentence structure and logical deductive reasoning to students, young and old alike. In any case, holding a conversation with your computer can be a lot of fun.

Let's see how the system accomplishes this and what its limitations are:

First, we have to get down to the basics of English. The system recognizes phrases grouped together in a sentence. A phrase is a group of one or more words that, together, convey a concept or identify an entity. These phrases can convey a subject, a verb, a preposition, a conjunction, a modifier or a question invoker. These represent the grammatical types that NLOS/1 can handle. You may ask - why not recognize individual words rather than phrases so that the conventional grammatical types - nouns, adjectives, verbs, adverbs, prepositions, conjunctions, interjections, and pronouns may be used? Well, recognizing nothing smaller than a phrase alleviates the problem of context usage. For example - "the President of the United States" contains adjectives, nouns and a preposition, yet the phrase identifies a single subject. Each grammatical type further conveys a type of information. Subjects identify a person (a who - "Tom", etc.), a place (a where - "New York City", etc.), or a thing (a what - "a banana", etc.).

Verbs convey the type of object clause they affect - in "Tom said nothing" and "Tom went home", etc., "said" acts on "what" and "went" acts on "where".

48 The 80 Notebook, Issue 04 (7/1980), S. 1, https://bit.ly/3CyoJ0C (zuletzt abgerufen am 03.07.2023). Hier wird ein Betriebssystem mit natürlichsprachlicher Ausgabe für den TRS-80-Computer vorgestellt, dessen BASIC-Listing zum Abtippen sich weiter hinten in der Ausgabe befindet. Die Druckqualität ist wie üblich schlecht; das Magazin wurde mit einem Matrixdrucker ausgedruckt und per Fotokopie vervielfältigt.

BASIC-Sourcecodes auf Papier wurden in Büchern (Handbücher zu Computern, Sekundärliteratur), Zeitschriften (Periodika, Club-Zeitschriften, Newslettern), als Computerausdrucke und handschriftlich zu Papier gebracht. Während Bücher und Zeitschriften, die durch ISBN und ISSN katalogisiert wurden, zumeist noch leicht auffindbar sind, lassen sich Publikationen der so genannten *Grauen Literatur*[49] weniger leicht in Archiven verorten. Hierzu zählen bereits die Handbücher der Computer, die oft von den Computerfirmen selbst verlegt und nur zusammen mit den Computern abgegeben wurden, aber auch von Amateuren publizierte Periodika (Club-Magazine, Newsletter, Vgl. Abb. 4). Die Bibliothekswissenschaft ist seit einiger Zeit bemüht, Katalogisierungssysteme für Graue Literatur zu entwickeln, damit die Bestände, die archiviert sind, auch erfasst und recherchiert werden können.

Noch problematischer sieht die Quellenlage bei so genannten *Ephemera* aus: Hierzu zählen sowohl handschriftlich notierte Sourcecodes[50] (Abb. 5) als auch Programmausdrucke (die nicht selten zum Zwecke des *Debuggings* ebenfalls mit handschriftlichen Marginalien/Annotationen versehen wurden, vgl. Abb. 6) und marginalisierte Printpublikationen. Es liegt im Wesen dieser Schriften, dass sie sich gar nicht systematisch erfassen lassen. Demzufolge ist die bibliothekswissenschaftliche Erforschung und Systematisierung solcher Quellen auch noch nicht weit

49 J. M. Gibb/E. Phillips: Bessere Zeiten für graue oder nicht herkömmliche Literatur. In: *Bibliothek* 3. 1979. Nr. 2, 122–126.

50 Hiervon gibt es im Heimcomputer-Zeitalter erstaunlich viele, was mit dem Käufer-/Nutzer-Alter der Computerbesitzer:innen zusammenhängt: Kinder und Jugendliche, die vielleicht nur sporadisch Zugang zu Computern hatten und Codes daher handschriftlich notiert haben, um sie später in den Computer eingeben zu können. Diese (Hand-)Schriften zählen zu den informatikdidaktisch interessantesten Quellen, weil sie das algorithmische Denken der Autor:innen ungefiltert vor Augen führen.

fortgeschritten[51], wenngleich erste Ansätze (auch zu Computercodes[52]) vorliegen.

Abb. 4: Deckseite des Newsletters vom TI-99-Computerclub in Salt Lake City (USA), Ausgabe 06/1988.

51 H. J. Jackson: *Marginalia. Readers Writing in Books.* New Haven/London: Yale Univ. Press 2001. Sowie: D. C. Greetham (Hg.): *The Margins of the Text.* Ann Arbor: The Univ. of Michigan Press 1997.

52 GreyNet International 1992–2017: *Document Types in Grey Literature. https://bit.ly/3NZfXpe* (zuletzt abgerufen am 29.06.2023).

Abb. 5: Handschriftlicher BASIC-Sourcecode mit Korrekturen[53]

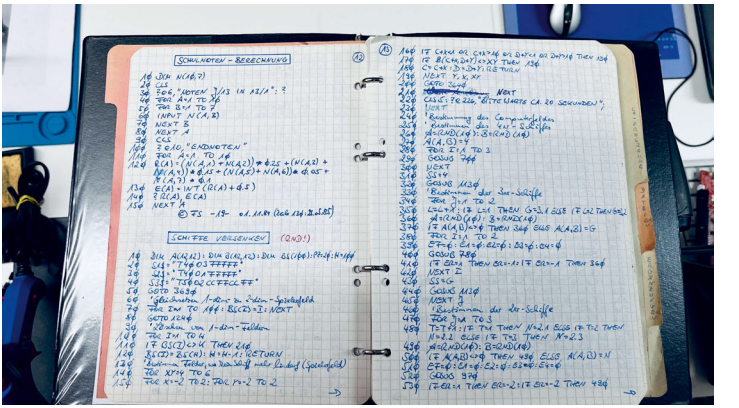
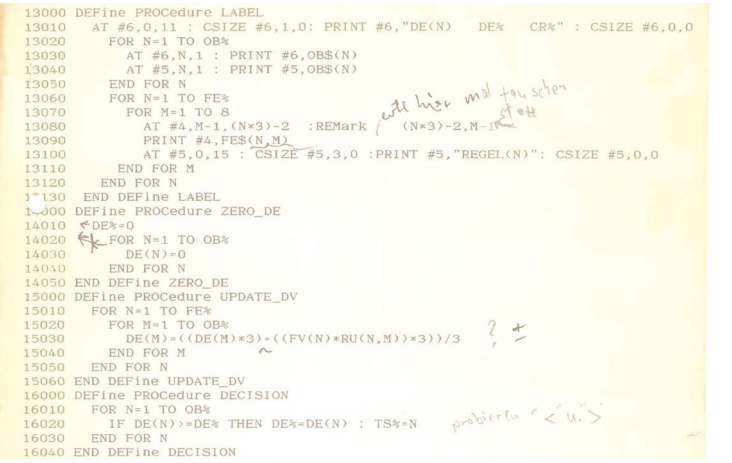


Abb. 6: Marginalisierter Programmausdruck (Auszug)[54]



```
13000 DEFine PROCedure LABEL
13010   AT #6,0,11 : CSIZE #6,1,0: PRINT #6,"DE(N)   DE%   CR%" : CSIZE #6,0,0
13020     FOR N=1 TO OB%
13030       AT #6,N,1 : PRINT #6,OB$(N)
13040       AT #5,N,1 : PRINT #5,OB$(N)
13050     END FOR N
13060     FOR N=1 TO FE%
13070       FOR M=1 TO 8
13080         AT #4,M-1,(N*3)-2  :REMark
13090         PRINT #4,FE$(N,M)
13100         AT #5,0,15 : CSIZE #5,3,0 :PRINT #5,"REGEL(N)": CSIZE #5,0,0
13110       END FOR M
13120     END FOR N
13130  END DEFine LABEL
14000 DEFine PROCedure ZERO_DE
14010   DE%=0
14020   FOR N=1 TO OB%
14030     DE(N)=0
14040   END FOR N
14050 END DEFine ZERO_DE
15000 DEFine PROCedure UPDATE_DV
15010   FOR N=1 TO FE%
15020     FOR M=1 TO OB%
15030       DE(M)=((DE(M)*3)+((FV(N)*RU(N,M))*3))/3
15040     END FOR M
15050   END FOR N
15060 END DEFine UPDATE_DV
16000 DEFine PROCedure DECISION
16010   FOR N=1 TO OB%
16020     IF DE(N)>=DE% THEN DE%=DE(N) : TS%=N
16030   END FOR N
16040 END DEFine DECISION
```

53 Quelle: eigenes Archiv.

54 Quelle: eigenes Archiv.

3. Schluss: Paperware und Knowledge Preservation

Die zuletzt aufgeführten Beispiele lassen den Reichtum der Quellen und die Bedeutung ihrer Inhalte vermuten, zeigen aber auch bereits die prekäre Situation, in der sich die *Paperware Preservation* befindet: Je ›grauer‹ und ephemerer ein Schriftgut ist, desto wahrscheinlicher ist es, dass es unwiederbringlich verloren gehen könnte. Einige der abgebildeten Dokumente stammen aus Sammlungen privater Computernutzer:innen (wie auch in Abb. 7), die als Paratexte zusammen mit ihren obsolet gewordenen Computern abgegeben wurden. Es ist nicht unwahrscheinlich, dass viele solcher Schriftstücke entsorgt wurden. Diejenigen, die der Forschung zur Verfügung stehen, müssen materialgereicht bewahrt werden. Hierzu zählt auch, eine bibliografische Systematik und Möglichkeiten für ihre Digitalisierung und Archivierung zu entwickeln.

Abb. 7: Textkonvolut aus dem Nachlass eines Computerhobbyisten[55]

Welches Wissen sich über die Programmierkulturen der Vergangenheit aus den Sourcecodes ziehen lässt, ist noch kaum abzuschätzen.

55 Quelle: eigenes Archiv.

Deutlich zeigt sich jedoch jetzt schon, dass die Sourcecodes nicht bloß Anweisungsketten für Computer sind, um Oberflächeneffekte in Form von Programmausgaben zu erzeugen, sondern in ihrer Unterschiedlichkeit, ihrer Individualität und ihren Beziehungen untereinander, in den individuellen Programmierstilen ihrer Autor:innen und den Publikationsformen und -medien auch komplexe Wissensspeicher darstellen. Sie lesbar zu machen – mit menschlichen und maschinellen Verfahren – öffnet der Kultur- und Technikgeschichte ab dem 20. Jahrhundert einen alternativen Blick auf den Werdegang der Digitalisierung und Algorithmisierung der Kultur. Überdies ermöglicht die philologische Erschließung der Textsorte Sourcecode interdisziplinäre Forschung zwischen den ›zwei Kulturen‹ technisch-informierter, Sourcecode-produzierender und linguistisch-informierter, quellenkritisch-lesender Disziplinen.