Quellcodekritik

Zur Propädeutik einer Verbindung zwischen analogen und digitalen Geisteswissenschaften

Markus Krajewski

1 Obskure Handlungen

Algorithmen beizukommen, erweist sich als komplizierte Angelegenheit, sei es, sie zu erhalten, sei es, sie zu lesen, sei es, sie zu verstehen, oder sei es, gegen sie vorzugehen, womöglich gar mit juridischen Mitteln. Man kann einen renitenten Nachbarn verklagen oder einem Wal den Prozess machen.[1] Aber ein Stück Code vor Gericht zu stellen, dessen Autorschaft keineswegs immer mit einer juristischen Person verknüpft ist, bedarf einigen Aufwands, um den Sachverhalt zu klären, Einblick in den Aufbau und Ablauf der Software zu gewinnen oder gar eine Veränderung des Algorithmus herbeizuführen.[2] Diese Schwierigkeit, die Funktionen eines Algorithmus zu durchschauen, einzelne Schritte

1 Vgl. David Graham Burnett: Trying Leviathan. The nineteenth-century New York court case that put the whale on trial and challenged the order of nature. Princeton 2007.

2 Wie das trotzdem gelingen kann, zeigt der Fall einer weißen, mittelalten, männlichen Person aus dem ländlichen Teil Finnlands, der aufgrund dieser Charakteristika von einem Algorithmus ein Mikrokredit versagt wurde, wogegen die Person klagte, vgl. Automating Society Report 2019, sowie Markus Krajewski: Hilfe für die digitale Hilfswissenschaft. Von den Digital Humanities verspricht man sich wahre Wunder, obwohl sie nur eine einfache Hilfswissenschaft sind. In: Frankfurter Allgemeine Zeitung, Nr. 85, N4, 2019, S. 119–121.

zu verstehen und ihrer Wirkungen habhaft zu werden, erscheint oft verkürzt zusammengefasst in einer weit verbreiteten, nichtsdestoweniger obskuren Metapher, mit deren Hilfe die Hilflosigkeit angesichts einer als übergroß erachteten Macht der Algorithmen zum Ausdruck gelangt: Die Rede ist dann von einer Black Box, innerhalb derer die Software ihre Funktionen entfaltet.[3] Etwas, ein bestimmtes Datum X, zum Beispiel der Satzanfang ›Ich bin …‹, gelangt in die lichtlose Kiste hinein, wird dort geschützt vor äußerer Wahrnehmung prozessiert, sodass kaum jemand weiß, wieso schließlich etwas Anderes, ein Datum Y, der vervollständigte Satz wie zum Beispiel ›… nicht Stiller‹, am Ende hinausgelangt.

Im Folgenden sei versucht, etwas Licht in diese Black Box zu bringen, um damit zugleich eine Methode vorzustellen, mit der insbesondere für Geisteswissenschaftler der Zugang zu Algorithmen erleichtert und verständlich zu werden verspricht.

Zunächst sei nach den verschiedenen Modi der Unzugänglichkeit gefragt, die eine Nachvollziehbarkeit der Codes verhindern, bevor in einem zweiten Schritt ein Verfahren vorgestellt sei, das als eine Art Gegenmittel der Opazität von Algorithmen entgegenwirken kann. Zuvor aber sei die Metapher der Black Box noch auf ihre historische Entwicklung und ihre sprachlichen Effekte hin ausgeleuchtet.[4]

In seiner historischen Herleitung der Black Box rekonstruiert Philipp von Hilgers zwei komplementäre Stiftungsmomente der Begrifflichkeit, ein materielles, an ein physisches Objekt gebundenes sowie ein ideelles, wo die Denkfigur im Kontext der jungen Kybernetik ihre Verwendung findet. Einerseits lässt sich die Vorstellung einer Black Box als technisches Gerät ohne direktes Verständnis seiner Funktionsweise

3 Exemplarisch, als regelrechte Gesellschaftsdiagnose, siehe Frank Pasquale: The Black box society. The secret algorithms that control money and information. Cambridge, MA 2016.

4 Vgl. Lawrence Lessig: Code. Version 2.0, 2. Aufl. New York 2006, und dagegen, mit einem differenzierenden Blick auf juridische Elemente von Computersystemen Cornelia Vismann/Markus Krajewski: Computer-Juridisms. In: Grey Room. Architecture, Art, Media, Politics 8, Nr. 29 (2007), Special Issue: »New German Media Theory«, S. 90–109.

recht präzise auf einen Botengang oder Technologietransfer datieren: Es kommt dem englischen Chemiker und Wissenschaftsmanager Henry Tizard zu, die Blaupausen und Schaltpläne eines neuartigen Radargeräts samt Prototypen aus britischer Entwicklung, das sog. Magnetron, im September 1940 an interessierte Forschungs- und Entwicklungsstäbe am MIT in Cambridge, MA, zu senden, und zwar in einer »black, metal deed box«, die das ebenso schwarz eloxierte Magnetron enthält, das dann zum Namensgeber für ein technisches Gerät wird, dessen Ein- und Ausgaben man kennt, deren technische Signalverarbeitung im Inneren aber obskur und undurchsichtig bleibt.[5] Die Metapher der Black Box nimmt also ihren Ausgang bei einer schwarzen Schachtel (Magnetron), die in einer schwarzen Schachtel transportiert wird, und deren Funktionsweise in diesem ersten Fall der Übertragung (von England in die USA) recht klar ist, auch wenn sich das Objekt seinerseits verschachtelt zeigt. »Die Black Box ist keine Kiste, die man nur zu öffnen braucht, um den Inhalt unverstellt zu erkennen. Sie enthält weitere Black Boxes.«[6]

Als Denkfigur etabliert sich der Begriff Black Box andererseits wenige Jahre später im Kontext der sich formierenden Kybernetik. Ausgangspunkt stellt dabei ein informelles Treffen am Institute for Advanced Study in Princeton am 6./7. Januar 1945 dar, anlässlich dessen unter den Teilnehmern, darunter Norbert Wiener und John von Neumann, rege von der Metapher Gebrauch gemacht wird. Warren McCulloch, einer der geistigen Väter der Artificial Intelligence, erinnert diese Szene, die er fälschlicherweise auf den Winter 1943/44 datiert,[7] als inspirierendes Rededuell zwischen von Neumann und Wiener. Beide debattieren leidenschaftlich über die mögliche Erkundung einer Kiste – in dem Fall

5 Philipp von Hilgers: Ursprünge der Black Box. In: Ana Ofak/Philipp von Hilgers (Hg.): Rekursionen. Von Faltungen des Wissens. München 2010, S. 135–153, hier S. 143–145.

6 Kathrin Passig: Fünfzig Jahre Black Box. In: Merkur. Deutsche Zeitschrift für europäisches Denken 71, Nr. 823 (2017), S. 16–30, hier S. 23.

7 Vgl. Warren McCulloch: Recollections of the Many Sources of Cybernetics. In: ASC Forum 6, Nr. 2, 1974, S. 5–16, sowie zur Korrektur dieses Datums Philipp von Hilgers: Ursprünge der Black Box, S. 150, Anm. 42.

eines von den Deutschen erbeuteten Kriegsgeräts –, um ihre Funktion zu erforschen, und zwar ohne die Schachtel zu öffnen. Eine Black Box, in die durch das Öffnen Licht fällt, gerät schließlich schnell zu einem Paradox, insofern ihr Inneres transparent wird und das Charakteristikum der Schwärze verschwindet.

Als ein Ergebnis der Überlegungen in Princeton sind nicht nur die Serie der Macy-Konferenzen ab dem darauffolgenden Jahr, sondern mittelbar ebenso die Dartmouth Conference im Sommer 1956 zu verbuchen, anlässlich derer die Entwicklung von künstlichen neuronalen Netzen auf den Weg gebracht wird. Die Metapher der Black Box besitzt ihren zweiten Ausgangspunkt also in einer Debatte über die Undurchsichtigkeit eines deutschen Kriegsbeuteguts, einer obskuren Kiste, deren genaue Funktionsweise sich nicht direkt, sondern nur über Umwege durch experimentelles Sondieren, durch variierendes Messen, durch eine Beobachtung der Eingangs- und Ausgangsdatenströme ermitteln lässt, insofern es zu verstehen gilt, wie genau das Gerät arbeitet und konstruiert ist.

Die prinzipielle Undurchsichtigkeit einer als Black Box bezeichneten Struktur besitzt hingegen auch Vorzüge. Von Hilgers weist darauf hin, dass für Informatiker und Software-Entwickler das Denkmodell der Black Box einige Abkürzungen, Effizienzsteigerungen und Vereinfachungen mit sich bringt. Solange die Schnittstellen klar definiert und Ein- wie Ausgabedaten damit eindeutig spezifiziert sind, bringt es erhebliche Zeitersparnis mit sich, wenn gerade *kein Blick* in die solcherart eingekapselten Methoden und algorithmischen Strukturen erfolgen muss. Was sich in mitgelieferten Programm- und Software-Bibliotheken verbirgt, was in seiner Verwendung klar definiert ist, darf hinsichtlich seiner Bauform verschlossen und hinsichtlich der internen Funktionen obskur bleiben, weil die Einkapselung in Software-Bibliotheken, -Repositorien und -Paketen als Stabilisierung oder Komplexitätsreduktion verstanden wird, deren Konstruktion nicht notwendigerweise offenzulegen ist. Das Schwarzgerät – so der *termi-*

nus technicus in Pynchons *Gravity's Rainbow* für eine Black Box[8] – dient demnach auch in der Softwareentwicklung weniger der Verschleierung, sondern vor allem der Vereinfachung und der Komplexitätsreduktion.

2 Schwarzgerät Durchschauen

Die Macht der Algorithmen resultiert aus einer Vielzahl an Undurchschaubarkeiten, die sich in verschiedenen Formen manifestieren. Von einer systematischen Durchmusterung dieses Feldes weit entfernt, sollen nun in einem zweiten Schritt mit Hilfe einer kurzen Heuristik zumindest drei Gründe unterschieden werden,[9] warum Algorithmen so schwer zu fassen sind.

2.1 Übersetzen

Eigentlich basieren Algorithmen als Problemlösungsverfahren auf einer Trias aus Funktionalität, Verständlichkeit und Eleganz.[10] Diese Grundprinzipien spiegeln sich nicht zwangsläufig in jedem Codefragment wider. Nicht nur, dass sich im Alltagsgeschäft zwischen Lösungsdruck und begrenzten Ressourcen die Codes nicht immer vorbildlich orientiert an diesen Vorgaben entwickeln lassen – Stichwort *quick and dirty*. Mehr

8 In diesem Fall entpuppt sich das geheimnisumwitterte Kriegsgerät als menschliche Ladung: der von Imipolex G ummantelte Gottfried, vgl. Thomas Pynchon: Gravity's Rainbow. New York 1973/1974.

9 Zu anderen Fallunterscheidungen, die den Einblick in die Black Box verhindern, sowie zu den damit verbundenen Gründen für die Unverständlichkeit von Software, also etwa allfällige Langsamkeit im Denken, Synchronisationsprobleme, Komplexität und Größe des Codes etc., siehe Passig: Fünfzig Jahre Black Box.

10 Donald E. Knuth: Computer Programming as an Art. In: Communications of the ACM 17, Nr. 12 (1974), S. 667–673, hier S. 670, der sich hier auf Jeremy Benthams utilitaristische Auffassung von Geschmack und Stil beruft, geht noch darüber hinaus, indem seine Programme schlicht auf Schönheit zielen; siehe zu Algorithmen als Geistestechnik auch Sybille Krämer: Operative Schriften als Geistestechnik. Zur Vorgeschichte der Informatik. In: Peter Schefe u.a. (Hg.): Informatik und Philosophie. Mannheim und Leipzig 1993, S. 69–83.

noch, in den überwiegenden Fällen entziehen sich die Algorithmen ihrer Lesbarkeit und damit einer prinzipiellen Nachvollziehbarkeit, weil nicht jede Software ihren Quellcode – also die Abfolge der einzelnen Befehle formuliert im berüchtigten »Klartext«[11] einer höheren Programmiersprache – zur Verfügung stellt. Nur sofern der Quellcode überhaupt vorliegt, lässt sich je nach verwendeter Sprache die Logik des Programms mit unterschiedlichem Aufwand entziffern, je nachdem, wie abstrakt sich die eingesetzten Kommandos, Datenstrukturen und Routinen zeigen: Sind sie in Englisch oder Mandarin verfasst? Folgen sie einer üblichen, auf Nachvollziehbarkeit angelegten Bezeichnungslogik? So ist ein Programmcode ohne allgemeinverständliche Befehlsbezeichnungen in seiner Abfolge kaum nachvollziehbar (vgl. Listing 1); sobald derselbe Code jedoch auf prosaische Bezeichnungen setzt, steht einer Erschließung des Algorithmus weniger entgegen (Listing 2).

Listing 1: Ein und derselbe Code in unterschiedlicher Stilistik, einmal abstrakt…

```
206  private long rhksog(int lmvlkmvlsdf) {
207    if (lmvlkmvlsdf <= 1)
208      return lmvlkmvlsdf;
209    return rhksog(lmvlkmvlsdf – 1) + rhksog(lmvlkmvlsdf – 2);
210  }
211
212  private long asdfasdfsadffasfsad(int herttjrtzhtr ) {
213    long gdgijimkfe = 0;
214    for (int dsfidsjfi = 0; dsfidsjfi <= herttjrtzhtr ; dsfidsjfi ++)
215      gdgijimkfe = rzsretsert ( herttjrtzhtr );
216    return gdgijimkfe;
217  }
```


11 Vgl. Geoffrey Winthrop-Young: Friedrich Kittler zur Einführung. Hamburg 2005, S. 59, 62ff.

Listing 2: … und einmal für Hermeneuten.

```
206  private long fibo(int n) {
207    if (n <= 1)
208      return n;
209    return fibo(n – 1) + fibo(n – 2);
210  }
211
212  private long fiboWithoutRecursion(int number) {
213    long j = 0;
214    for (int i = 0; i <= number; i++)
215      j = addFibonacciS(i);
216    return j;
217  }
```

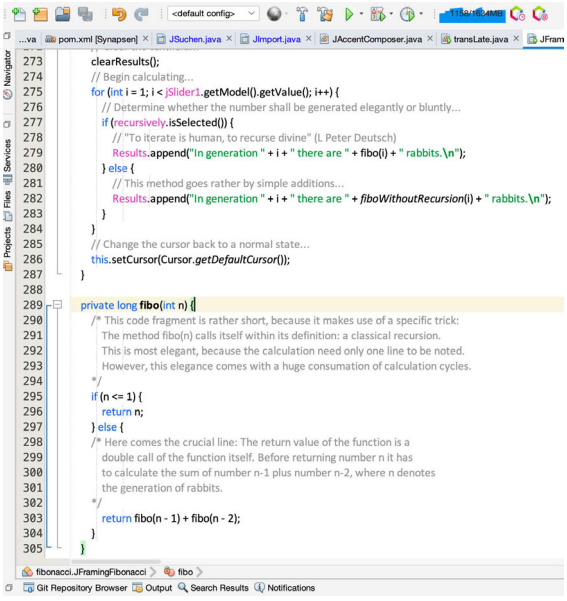

In einem zweiten Schritt, dem Kompilieren, also durch die Übersetzung des Quellcodes in den von der Maschine ausführbaren *binary code*, wird der Algorithmus noch weiter verunklart (*obfuscated*) und damit in der Black Box namens Binärcode versiegelt, die sich anschließend nicht ohne Weiteres wieder öffnen lässt.[12] Durch die Übersetzung in einen vom Betriebssystem ausführbaren Code (durch Umwandlung in eine *.bin oder *.exe-Datei) werden die Algorithmen transformiert, verschlossen in einer Schachtel, und entziehen sich damit einer weiteren Möglichkeit, sie einzuordnen.

Es gibt ein Pendant im Juridischen zum Schließen dieser Codes, das auf eine alte Rechtspraxis im Römischen Recht des 6. Jahrhunderts zurückgeht: die Digesten, bis heute eine der Grundlagen des bürgerlichen Rechts, die als eine Materialsammlung, als eine Kompilation verschiedener Rechtsfälle und Kommentare ein Gedächtnis des Urteilens und Entscheidens boten und zugleich als Regelwerk dienten. Auf Geheiß von Kaiser Justinian wird 533 AD aus dieser Materialsammlung ein Buch, ein

12 Joasia Krysa/Grzesiek Sedek: Source Code. In: Matthew Fuller (Hg.): Software studies. A lexicon. Cambridge, MA, und London 2008, S. 232–243; Markus Krajewski: Against the Power of Algorithms. Closing, Literate Programming, and Source Code Critique. In: Law Text Culture 23 (2019), S. 119–133, hier S. 123.

Kodex gemacht, das die Ansichten ausgewählter römischer Juristen der Republik- und Kaiserzeit bündelt und mit Gesetzeskraft ausstattet, wobei ein Großteil der ausufernden Kommentare außen vor bleibt. Indem sich die Buchdeckel schließen, kodifizieren sie ihren Inhalt, das heißt sie erheben ihn zum geltenden Recht. Nichts anderes bedeutet Code: Schließen. Die Kodifizierung leitet sich vom Schließen der Buchdeckel ab. Die Kommentare aber bleiben außen vor als (nicht-legitimes) Beiwerk und Erläuterung ohne Gesetzeskraft.[13]

Abbildung 1: Ein Java-Programm, hier im Original …

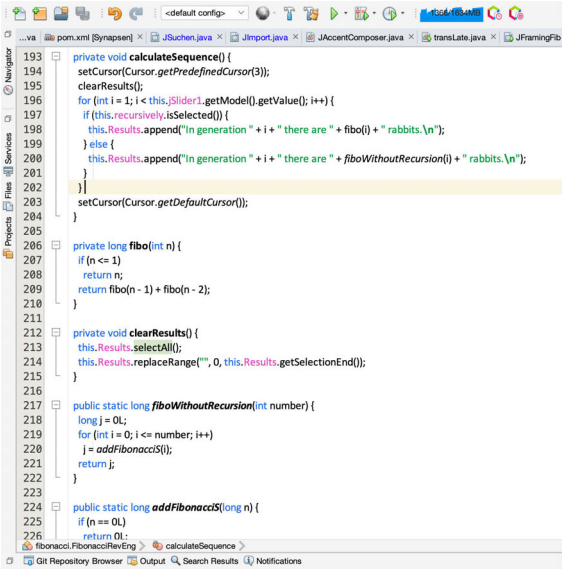


© Markus Krajewski

13 Vgl. Markus Krajewski/Cornelia Vismann: Kommentar, Code und Kodifikation. In: Zeitschrift für Ideengeschichte 3, Heft 1 (2009), Themenheft »Kommentar«, S. 5–16, hier S. 7f.

Der Vorgang der Schließung oder Kodifizierung ist aber keineswegs irreversibel. So wie das Römische Recht auch von Zeit zu Zeit einer Erneuerung und Modifikation bedarf, lässt sich mit Hilfe einiger Werkzeuge die Übersetzung in unlesbare Maschinensprache zumindest teilweise wieder rückgängig machen. Auch ausführbarer Code kann, nicht zuletzt weil es sich beim Computer um eine deterministische Maschine handelt, wieder in seinen Ausgangszustand zurückversetzt werden. Diese Praktik wird als *reverse engineering* bezeichnet und führt das Beispiel aus Listing 1 in den vergleichsweise wohlgeordneten Zustand von Abb. 1–2 zurück.

Abbildung 2: … und unten reverse engineered



© Markus Krajewski

Auffällig dabei ist, dass der Code ist nicht mehr so gut lesbar scheint wie zuvor, weil er gleichsam ›nackt‹ ist. Die Kommentare und erläutern-

den Strukturen fehlen, es sind allein die funktionalen Elemente, also die Methoden und Datenstrukturen, wieder hergestellt, nicht aber die Kommentare.

2.2 Strömen

Eine zweite Unzugänglichkeit der Algorithmen ergibt sich vor allem aus den neueren Infrastrukturen digitaler Welten: Die Infrastrukturen des Internet prozessieren vornehmlich Datenströme. Dateien und damit auch ausführbare Codes sind nicht mehr zwangsläufig vollständig lokal vorhanden, sondern in Wolken oder externe Speicher verlagert, aus denen sie nach Bedarf abgerufen, das heißt in Form eines Datenstroms lokal verfügbar gemacht werden. Dabei handelt es sich – die Wolkenmetapher vernebelt zu Recht – um ein ephemeres Verfahren, weil es keinen expliziten, für den Nutzer transparenten Speicherort für diesen Datenstrom lokal vorsieht. Was aber keinen Ort hat, bleibt unverfügbar, unadressierbar, und damit obskur. Ohne Adresse kann man eines Datums nicht habhaft werden. Die Algorithmen sind einmal mehr eingekapselt in Datenströme, die nicht mehr vollständig vorliegen, sondern in Echtzeit übertragen ihre Wirkungen lokal entfalten und danach schon wieder vergehen. Schon in den Fragmenten der Vorsokratiker ist die Schwierigkeit benannt: Man steigt niemals in denselben Fluss.[14] Das grundlegende Problem bleibt auch beim Umgang mit Datenströmen bestehen, und dieses Problem ist der Anfang: Wo ist zu beginnen, will man einen Datenstrom verstehen, wenn man nicht weiß, was überhaupt am Anfang stand? Ohne Anfang aber kann kein Programm beginnen, ausgeführt, geschweige denn in seinen Ausgangszustand zurückversetzt werden.

14 Jaap Mansfeld: Die Vorsokratiker. Griechisch/Deutsch, erw. Neuausg. Stuttgart 2012, Fragment 91, vgl. auch 12, 49a.

2.3 Schichten

Eine ganz anders gelagerte Form der Unzugänglichkeit von Algorithmen zeigt sich in dem, was gegenwärtig die Debatten der *computer science* über soziotechnische Folgen auch über das Fach hinaus in besonderem Maße bestimmt: Die Künstlichen Intelligenzen und ihre sozialen, habituellen, ethischen, ökonomischen und nicht zuletzt juridischen Effekte. Die grundlegende Schwierigkeit liegt – anders als bei herkömmlichen Software-Architekturen und -Systemen – darin, dass auch die verständigsten Informatiker nicht mehr erklären können, wie genau Entscheidungen innerhalb von Large Language Models (LLM) getroffen werden. Mehr noch: Die Ergebnisse erweisen sich weder auf Mikro- noch auf Makroebene als systematisch reproduzierbar. Der Grund liegt in der Architektur der LLM, die als elektronische Simulation menschlicher Gehirne ein Speichersystem aus künstlichen Neuronen – genauer: Perzeptronen – darstellen. Dieser Speicher muss mit Daten – seien dies Bilder, Texte, Stimmen – so lange *trainiert* werden, bis ein bestimmtes Wissen über die Genera und die Spezifika dieser Eingabedaten gebildet worden ist. Dieses Wissen wird dabei rein technisch in numerischen Vektorräumen gespeichert, und zwar als Wahrscheinlichkeiten einzelner Netzknoten, mit denen ein Eingabesignal auf die damit verbundenen anderen Netzknoten reagiert. Nach Abschluss des Trainings kann dieses sog. Modell seinerseits auf Nutzeranfragen reagieren. Auf spezifische Eingaben antworten diese Modelle jedoch – unabhängig von einer zuvor eingestellten ›Temperatur‹ – nicht immer gleich, sondern jeweils anders und unvorhersehbar, weil die Entscheidungspfade durch die jeweiligen Schichten der künstlichen Neuronen nach probabilistischen Kriterien gewählt werden und nicht starren Wegvorgaben nach einer Wenn-Dann-Struktur folgen, also nicht mit fest verdrahteten Gewissheiten operieren operieren. Die Systemarchitektur folgt einem »konnektionistischen Paradigma«,[15] in dem sich

15 Hannes Bajohr: Algorithmic Empathy: On Two Paradigms of Generative Digital Literature. In: bmc:ct. Working Papers 1, Nr. 2 (2020), DOI: 10.12685/bmcct.2020.004, S. 16ff.

die einzelnen künstlichen Neuronen – salopp gesagt: mal so, mal so – zusammenschließen zu jeweils unterschiedlichen Konstellationen. Das heißt, zweimal dieselbe Eingabe in ein LLM bzw. in eine Black Box wird selbst bei prompt aufeinander folgenden Prozessen auch unterschiedliche Ergebnisse zeitigen. Um die Analogie zu juridischen Verfahren einmal mehr zu bemühen: Man könnte ein LLM als riesiges, geschichtetes, verschlossenes Digest verstehen: Darin ist eine Vielzahl an Fällen, Urteilen und rechtlichen Prozessen abgelegt, allerdings nicht im Wortlaut oder systematisch geordnet mit Register und Index, sondern wild durcheinander, fragmentiert, jedes Bruchstück mit unterschiedlicher Gewichtung bezüglich seiner Relevanz notiert, wenn es um die Frage geht, wie es mit anderen, benachbarten Partikeln und wiederum mit deren Nachbarn zusammenhängt. Folgt man dieser Analogie, so offenbart sich ein Gerechtigkeitsproblem, denn jede (wortgleiche) Eingabe liefert mithin ein anderes Ergebnis, infolge der statistisch voneinander abhängenden Mikroentscheidungen an jedem Knotenpunkt der verborgenen Netzwerkschicht, die in der Summe zu einem jeweils anderen Urteil führen können.

Die Architektur eines LLM, mit dem das sog. *deep learning* erfolgt (dabei kommen rekurrente, also mit sich selbst und mit vorhergehenden und nachgelagerten Schichten verknüpfte neuronale Netzwerke zum Einsatz, vereinfacht gesagt: Architekturen mit Fehlerkorrektur),[16] eine solche Architektur besteht nicht nur in ihrer Handhabung aus einer Black Box. Ein typisches LLM ist – in der Tradition seines Grundbausteins, dem Perzeptron – aus drei Schichten aufgebaut: (1) der Eingabeschicht, (2) dem *hidden layer* sowie (3) der Ausgabeschicht. Und dieser *hidden layer*, der aus zahlreichen, miteinander verknüpften Schichten künstlicher Neuronen besteht,[17] stellt gleichsam – wie in ihrer Urszene, dem Transport des schwarzen Magnetrons nach Massachusetts in einer schwarzen Box – die Black Box in der Black Box dar.

16 Vgl. Ethem Alpaydın: Machine learning. The new AI. MIT Press essential knowledge. Cambridge, MA 2016, S. 85ff.

17 Pedro Domingos: The master algorithm. How the quest for the ultimate learning machine will remake our world. New York 2015, S. 101.

Das Wissen eines LLM ist also nicht allein verkapselt in verschiedene schwarze Schachteln, sondern zudem darin noch *zerstäubt*, numerisch dispergiert in Vektorräumen, verteilt an eine Vielzahl kleinster elektronischer Speicherelemente (künstlicher Neuronen), verbunden mit nichts als einem statistischen Wert als Übergangswahrscheinlichkeit zu ihren jeweiligen Nachbarn und Nachbarn der Nachbarn. Im Kern der verkapselten Blackbox eines künstlichen neuronalen Netzes arbeitet ein Nebel, eine partikularisierte Form zufälliger Mikroentscheidungen, die in ihrer Fragmentarisierung nichtsdestotrotz immer beachtlichere Leistungen erbringen.[18]

Wenn nun aber selbst Computerwissenschaftler nicht mehr die Funktionsweise ihrer eigenen, immer noch auf deterministischen Maschinen operierenden Systeme nachvollziehen können, wie soll da eine kritische Perspektive auf diese Technologie im Detail möglich sein? Das Problem ist freilich auch seitens der Informatik längst erkannt und findet unter dem Etikett der ›Explainable AI‹ (XAI) gesteigerte Aufmerksamkeit.[19] Eine zentrale Erkenntnis dürfte sein, dass sich die Black Box im Prozess ihres Operierens prinzipiell nicht mehr öffnen lässt. Nur auf der Ebene der Konzeption lässt sich noch Transparenz herstellen, was wiederum auf die Ebene der Quellcodes sowie der Skripte, mit denen LLM gebaut und trainiert werden, zurückführt. Nur hier, auf dieser konzeptionellen und zugleich konkreten Ebene der Umsetzung – vor dem eigentlichen Ausführen des Codes – lassen sich die Wege der Entscheidungsfindung annäherungsweise mit Transparenz nachvollziehen. Doch die Arbeit am Quellcode bedarf ihrerseits einiger Vorbereitungen. Der Weg zur Einsicht folgt, ganz klassisch, den Wegmarken einer philologisch-historiografischen Tugend: der Quellenlektüre. Was heißt das konkret?

---

18 Vgl. etwa für das Visuelle Emily Lanza: Who Painted Rembrandt? Copyright and Authorship of Two Rembrandt Portraits, 2018, und für das Textuelle die Leistungen von Claude, Llama, Gemini, GPT-4, GPT-5.x usw., entworfen von Tom Brown u.a.: Language Models are Few-Shot Learners, arxiv.org, 2020.

19 Vgl. etwa Wojciech Samek u.a.: Explainable AI: Interpreting, Explaining and Visualizing Deep Learning. Cham 2019.

3 Remedium Quellcodekritik

Auch frei zugänglicher Code entzieht sich für gewöhnlich einer einfachen Lesbarkeit. Selbst wenn die Quellen ohne Einschränkung einsehbar sind und keine weitere *obfuscation* oder *nebulization* vorliegt, zeigt sich Code im Umgang von anderen Lesern als *geeks*, *hackers* und *nerds* widerständig. Und doch ist es notwendiger denn je, dass jenseits dieser Gruppen und ihrer genuinen *computer literacy*, also auch in den Geisteswissenschaften und benachbarten Disziplinen wie der Jurisprudenz oder den Sozialwissenschaften die Fähigkeit beherrscht, vermittelt und erforscht wird, neben komplexen philosophischen, juridischen und literarischen Texten ebenso Code zu lesen und zu verstehen, und zwar aus einer kritischen Perspektive.

An diesem Punkt setzt die ›Kulturtechnik Programmieren‹ an, um eine neue Methodologie für den Umgang mit Code jenseits der *computer science* vorzuschlagen. Dabei gilt es, einem Verfahren zu folgen, das unter dem Begriff Quellcodekritik[20] darauf zielt, den Code eines Softwareprojekts zu einem extensiven, erläuternden Kommentar zu verdichten. Im besten Fall stellt das Ergebnis dieses Verfahrens ein gemeinsames Werk aus Text und Code bereit, eine Kulturgeschichte des Codings in Buchform und zugleich als Software Application.

Welche Funktion kommt dabei der Kulturtechnik Programmieren zu? Kulturtechniken sind darauf ausgelegt, im Zusammenspiel von gezielten körperlichen Gesten und dem Gebrauch von Hilfsmitteln wie Werkzeugen, Instrumenten oder anderen medialen Objekten eine Handlung zu vollziehen, die in spezifischer Weise eine kulturelle Wirksamkeit entfaltet. Die Kulturtechnikforschung untersucht entsprechend die medialen Praktiken dieser kulturstiftenden Prozesse, verbunden mit den daran beteiligten Verfahren, Gesten und Werkzeugen, in ihrer historischen Entwicklung und in ihren kultur- und

20 Vgl. dazu ausführlich Hannes Bajohr/Markus Krajewski (Hg.): Quellcodekritik: Zur Philologie von Algorithmen. Berlin 2024.

erkenntnistheoretischen Grundlagen.[21] Dabei umfassen Kulturtechniken immer auch eine ästhetische Komponente, die über die reine Funktionalität hinaus – ganz wie es Donald Knuth für das Programmieren fordert[22] – ebenso Aspekte des Stils, Fragen von Eleganz und nicht zuletzt das Charakteristikum der Schönheit einbeziehen.

Trotz der großen Aufmerksamkeit und einer wachsenden internationalen Beachtung der Kulturtechnikforschung[23] sind Fragen des Digitalen, etwa zur Algorithmen-Entwicklung und ihrer kulturtechnischen Funktionsweise, bis dato weitestgehend unberücksichtigt geblieben. Diesem Desiderat gilt es zu begegnen, indem nach dem Coding als Kulturtechnik zu fragen ist, nach der Fähigkeit also, Programmcode nicht nur zu lesen und zu schreiben oder zu entwickeln für die alltägliche Software-Produktion, sondern ihn ebenso diskursivieren, historisieren und kritisch einordnen zu können. Dieser Fähigkeit kommt für Geisteswissenschaftler künftig eine Schlüsselstellung zu, denn es braucht eine derartige Kompetenz auch jenseits der professionellen Softwareentwickler, Computeringenieure, Interface-Designer und nicht zuletzt jenseits der selbstlernenden Maschinen, um deren Handeln und Entwerfen, deren Erfinden und Abrichten der Algorithmen mit kritischer Reflexion und umfassendem Verständnis zu begleiten. Nur so kann es gelingen, die oft beschworene ›Macht der

21 Vgl. Erhard Schüttpelz: Die medienanthropologische Kehre der Kulturtechniken. In: Archiv für Mediengeschichte 6 (2006), S. 87–110; Bernhard Siegert: Cultural Techniques: Or the End of the Intellectual Postwar Era in German Media Theory. In: Theory, Culture & Society 30, Nr. 6 (2013), S. 48–65; Bernhard Siegert: Cultural Techniques. Grids, Filters, Doors, and other Articulations of the Real. New York 2015.

22 Vgl. Anm. 10 auf S. 101.

23 Vgl. exemplarisch Geoffrey Winthrop-Young: The Kultur of Cultural Techniques. Conceptual Inertia and the Parasitic Materialities of Ontologization. In: Cultural Politics 10, Nr. 3 (2014), S. 376–388, Geoffrey Winthrop-Young: Discourse, Media, Cultural Techniques: The Complexity of Kittler. Modern Language Notes 130, Nr. 3 (2015), S. 447–465, Geoffrey Winthrop-Young: Siren Recursions. In: Stephen Sale/Laura Salisbury (Hg.): Kittler Now: Current Perspectives in Kittler Studies. Cambridge 2015, S. 71–94, Geoffrey Winthrop-Young: The Kittler Effect. In: New German Critique 44, Nr. 132/3 (2017), S. 205–224.

Algorithmen‹ zu entzaubern, sie einzuhegen und zu analysieren. Nur indem man Codes kommentiert, lassen sie sich domestizieren.[24] Konkret bedeutet dies für Textwissenschaftler, eine kritische Distanz zu Algorithmen einnehmen zu lernen, sie also zu decodieren, zu verstehen, zu zerlegen, um sie auf ihre Funktionsweise und Wirkungen, ihre sprachlichen Eigenheiten, ihre Machart, ihren Stil hin zu überprüfen und sie nach ihren historischen, juridischen, kulturellen oder politischen Vorgaben und Effekten einordnen zu können. Dieser Ansatz wird mit etwas veränderter Stoßrichtung von den *Software Studies* verfolgt, wie sie Matthew Kirschenbaum, Stephen Ramsay, Lev Manovich oder Nick Montfort propagieren,[25] nicht zuletzt um damit ebenso die *digital literacy* für Geisteswissenschaftler zu stärken.[26] Insbesondere Mark Marino, der seit einigen Jahren das innovative Feld der *Critical Code Studies* entwickelt,[27] konnte etwa anhand eines *close reading* der kollaborativ verfassten Novelle *exquisite_code*[28] oder auch anhand von Friedrich

24 Krajewski: Against the Power of Algorithms.

25 Matthew Kirschenbaum: Hello Worlds. Why humanities students should learn to program, 2009, https://mkirschenbaum.wordpress.com/2010/05/23/hello-worlds/(23.2.2024), Ders.: What Is Digital Humanities and What's It Doing in English Departments? In: ADE Bulletin 47, Nr. 150 (2010), S. 55–61, Stephen Ramsay: Reading machines. Toward an algorithmic criticism. Urbana 2011, Lev Manovich: Software takes command. New York 2013, oder Nick Montfort: Exploratory programming for the arts and humanities. Cambridge, MA 2016.

26 Vgl. dazu auch Geoff Cox/Christopher Alex McLean: Speaking code. Coding as aesthetic and political expression. Software studies. Cambridge, MA 2013; David Berry/Anders Fagerjord: Digital Humanities. Knowledge and Critique in a Digital Age. Oxford 2017; Daniel Punday: Computing as Writing. Minneapolis 2015, chap. 3.

27 Mark C. Marino: Critical Code Studies. electronicbookreview.com, 2006, und Ders.: Critical Code Studies and the electronic book review: An Introduction, electronicbookreview.com, 2010.

28 Mark C. Marino: Reading »exquisite_code«. In: N. Katherine Hayles/Jessica Pressman (Hg.): Comparative textual media. Transforming the humanities in the postprint era. Minneapolis 2013, S. 283–309.

Kittlers Codeproduktion zeigen, wie eine diskursanalytische Lesart von Algorithmen funktioniert.[29]

Listing 3: A recursion generating Fibonacci numbers and its explanation

```
private void calculateSequence() {
    /* What seems so interesting about calculating this series of numbers?
    There are two reasons, one is the history of this algorithm, the other is
    the style, how this algorithm can be implemented.

    1. In 1202, Fibonacci published the Liber Abaci, introducing not only the arabic
       numerals, but also algorithms to occidental mathematics. It contains a famous series of
       numbers, baptised after him. He develops his example by a famous scenario:

       "How Many Pairs of Rabbits Are Created by One Pair in One Year.
    A certain man had one pair of rabbits together in a certain enclosed place, and one wishes to know [...]."

    2. The sequence of numbers can be calculated in two different way, on the one hand most elegantly
       by a recursive method which starting with the last generation and going back to the first, while it
       calls itself as a method while it calls itself as a method while it calls itself as ...
       until the initial value n=1 is reached. Then all the numbers are available and can be added to the final
       sum.
       On the other hand, the value can be calculated just by remembering the last and the last but one value.
       This is rather easy concerning calculating power and ressources, however it takes more code to
       produce the non-recursive algorithm.

    In the following, both ways are implemented, so the user can compare the differences by walking the
    code line by line ...
    */

    // Change the cursor to a wait state ...
    this.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
    // Clear the textfield ...
    clearResults();
    // Begin calculating ...
    for (int i = 1; i < jSlider1.getModel().getValue(); i++) {
        // Determine whether the number shall be generated elegantly or bluntly ...
        if (recursively.isSelected()) {
            // "To iterate is human, to recurse divine" (L Peter Deutsch)
            Results.append("In generation " + i + " there are " + fibo(i) + " rabbits.\n");
        } else {
            // This method goes rather by simple additions ...
            Results.append("In generation " + i + " there are " + fiboWithoutRecursion(i) + " rabbits.\n");
        }
    }
    // Change the cursor back to a normal state ...
    this.setCursor(Cursor.getDefaultCursor());
}

private long fibo(int n) {
    /* This code fragment is rather short, because it makes use of a specific trick:
       The method fibo(n) calls itself within its definition: a classical recursion.
       This is most elegant, because the calculation need only one line to be noted.
       However, this elegance comes with a huge consumation of calculation cycles.
    */
    if (n <= 1) {
        return n;
    } else {
    /* Here comes the crucial line: The return value of the function is a
       double call of the function itself. Before returning number n it has
       to calculate the sum of number n-1 plus number n-2, where n denotes
       the generation of rabbits.
    */
        return fibo(n - 1) + fibo(n - 2);
    }
}
```

© Markus Krajewski

29 Mark C. Marino: Critical code studies. Cambridge, MA 2020, S. 161–197.

Statt also literarische Texte quantitativ auf Statistiken oder numerische Eigenheiten wie etwa Worthäufigkeiten zu untersuchen, ein Verfahren, das allzu häufig in den Digital Humanities zur Anwendung gelangt,[30] geht es der Kulturtechnik Programmieren darum, die geisteswissenschaftlichen Kernkompetenzen der kritischen Lektüre zu übertragen auf ›das Digitale‹, was nicht nur heißt, Algorithmen verstehen, einschätzen und weiterschreiben zu können, sondern, sie mit den Termini und Theorien, den Denkfiguren und Dispositiven einer literarischen Kritik zu fassen und zu kommentieren. Erst diese Form der Kontextualisierung erlaubt es, Code zu dekonstruieren, als wäre er ein literarischer Text.

Im Gegensatz zu diesem neuen Ansatz ist die umgekehrte Idee, Softwareentwicklung auf literarischem Niveau zu betreiben, keineswegs neu. Donald E. Knuth, Autor der epochalen *Art of Computer Programming* (1968–2025) und des Schriftsatzsystems TEX, veröffentlichte bereits 1984 unter dem mehrdeutigen Titel ›Literate Programming‹ den Vorschlag, Quellcode gleich so zu schreiben, dass er nicht nur die Befehle in der jeweiligen Programmiersprache beinhaltet. Vielmehr sollten die einzelnen Anweisungen und Programmstrukturen vom Entwickler zugleich auf einer Meta-Ebene intensiv beschrieben und kommentiert werden.[31] Der Quellcode enthielte damit die einzelnen Befehle und Datenstrukturen zugleich mit deren Dokumentation. Auf diese Weise wären Algorithmen transparent und blieben – nicht nur für ihre Autoren, sondern auch für spätere Leser und Bearbeiter – leichter nachvollziehbar (siehe Listing 3). Kaum nötig zu erwähnen, dass dieses Paradigma im Alltag einer professionellen Softwareentwicklung kaum Anwendung findet.

Literatur, Code und deren Kritik hängen strukturell enger zusammen, als es zunächst scheinen mag. So wie die Kulturtechnik Lesen auf akademischem Niveau in die Lage versetzt, die Machart von Texten, die Rhetorik, die Produktion von Affektzuständen, den Stil und

30 Exemplarisch: Franco Moretti: Distant reading. London 2013.

31 Donald E. Knuth: Literate Programming. In: The Computer Journal 27 (1984), S. 97–111.

die literarischen Denkfiguren und -bezüge offenzulegen, so versetzt die *coding literacy* in eine Position, dem Unterworfensein gegenüber algorithmischen Strukturen und softwaretechnisch implementierten Abhängigkeiten abzuhelfen, die Black Box namens Code zu öffnen und den Blick auf die Konstruktionsform der Algorithmen freizulegen.[32] In diese Lektüren spielen auch Fragen der Machart, der Poetologie von Algorithmen hinein. Die Lösung eines informatischen Problems kennt schließlich viele Wege, von denen manche der Langeweile einer zwölfspurigen Autobahn durch die Bay Area an der amerikanischen Westküste, andere dem Entdeckungspotenzial des Feldwegs aus Arno Schmidts *Zettels Traum* und wieder andere, wenngleich selten, dem noch zu findenden Pfad gleichen, den Isak in Knut Hamsuns *Segen der Erde* in die Wildnis schlägt, um aus ihr eine zivilisatorische Zukunft zu formen. Mit einem Wort, bei den Einordnungen und hermeneutischen Lektüren von Code gilt es nicht zuletzt, auf Intentionen, Formulierungen und Stilfragen zu achten. Denn nicht zuletzt um zu erkennen, wie ein Nutzer von der Software benutzt wird, bleibt es notwendig, die Machart und die Bauweise der Algorithmen entschlüsseln, nachvollziehen und kritisch wenden zu können.[33]

Das Verfahren zur kritischen Reflexion von Algorithmen und zugleich zur konsequenten Stärkung der *coding literacy* in den Geisteswissenschaften auf der grundlegenden Ebene der Codes lässt sich konzeptuell unter dem Begriff ›Quellcodekritik‹ fassen. Diese vereint die klassische Gelehrsamkeit einer historischen Quellenkritik[34] – die sich einer sorgfältigen Analyse und Aufbereitung des Ausgangsmaterials, hier also der Algorithmen, befleißigt – mit einer theoriegesättigten, an praktischer Funktionalität ausgerichteten, kritisch kommentierenden Lesart von Programmstrukturen. Konzeptuell beinhaltet diese Methode einerseits den Bereich der Quellcodeerschließung, also den Zugriff auf

32 Annette Vee: Coding literacy. How computer programming is changing writing. Cambridge, MA 2017.

33 Markus Krajewski: Hilfe für die digitale Hilfswissenschaft.

34 Daniela Saxer: Die Schärfung des Quellenblicks. Forschungspraktiken in der Geschichtswissenschaft 1840–1914. München 2014, S. 376–384.

die Algorithmen in *open source repositories* wie github oder, ungleich komplizierter, mit Verfahren des Dekompilierens und des *reverse engineering*. Andererseits bedeutet Quellcodekritik eine kritische Lektüre der Codes, die in ihrer Dynamik und Veränderbarkeit wie eine historische Quelle aufzufassen sind, die nicht zuletzt wegen ihrer mitunter zahlreichen Versionen[35] einer weiteren Einordnung und Kommentierung bedarf.

4 Macht der Kommentare

Insbesondere das Kommentieren stellt dabei die zentrale mediale Praktik der Kulturtechnik Programmieren dar. Schon seit den klassischen Verwendungsformen des Kommentars sowohl in der theologischen Auslegung als auch in der juridischen Praxis seit der Spätantike gilt: Kommentare stützen den Text, sie bedingen ihn, machen ihn überhaupt erst haltbar. Sei es eine Heilsschrift, sei es ein Gesetz, Kommentare bewahren den Text davor, zu erstarren oder unverständlich zu werden, sie halten Argumente fluide in der Diskussion, indem sie einzelne Aussagen hervorheben und diskursivieren.[36] Eine ähnliche Funktion besitzt der Kommentar in der textphilologischen Arbeit, der Editionstätigkeit und der *critique génétique*: Unklare oder mehrdeutige Stellen, Varianten und Streichungen im Original werden durch Hinweise und Einordnungen erläutert, um so die Genese wie die Bauform des Textes transparent zu machen.[37] Und nicht zuletzt zeigt sich die Notwendigkeit des Kommentierens auch im Rahmen einer digitalen Quellenphilologie, die

35 Vgl. Markus Krajewski: branch, diff, merge. Versionskontrolle und Quellcodekritik. In: Jörg Paulus/Andrea Hübener/Fabian Winter (Hg.): Duplikat, Abschrift & Kopie. Kulturtechniken der Vervielfältigung. Wien und Köln 2020, S. 21–40.

36 Vismann/Krajewski: Computer-Juridisms, S. 102; Krajewski/Vismann: Kommentar, Code und Kodifikation, S. 5–9.

37 Gérard Genette: Paratexte. Das Buch vom Beiwerk des Buches [1992]. Frankfurt a.M. und New York 1997; Almuth Grésillon: Literarische Handschriften. Einführung in die critique génétique. Bern 1999.

editionsphilologische Verfahren auch auf Software als Untersuchungsgegenstand anwendet.[38]

Im Kontext der Methodik einer Kulturtechnik Programmieren umfasst ein Kommentar all diese Funktionen. Allerdings geht die Methode noch einen entscheidenden Schritt weiter, um den Kommentar in seiner gesamten erkenntnisstiftenden Breite zur Entfaltung zu bringen. Eine der hervorragenden epistemischen Eigenschaften des Kommentars liegt im systematischen Wechsel der Ebenen: Zwischen Text und Kommentar besteht immer schon eine kleine Kluft, mit deren Überbrückung gleichsam automatisch ein Perspektivwechsel einhergeht, verbunden mit einem notwendigerweise distanzierten Blick von der Kommentarebene auf das Eigentliche, sei dies ein literarischer Text oder sei es Code. Dieser Perspektivwechsel birgt ein Moment der Reflexion auf das eigene Tun, womit dem Kommentator ein kritischer Blick auf die Praktiken des eigenen Schreibens im Vollzug möglich ist. Mit ihrem systematischen Ebenenwechsel bilden Kommentare den epistemologischen Hebel für eine beständig oszillierende Perspektive auf das Geschriebene und damit ein noch unausgeschöpftes Erkenntnisinstrument. Denn Kommentare dienen immerzu als unscheinbare Assistenten der Reflexion, indem sie zur Erläuterung und Plausibilisierung, zur Auslegung und Verknüpfung mit anderen Texten einladen und provozieren. Das Konzept der Quellcodekritik nutzt diese oszillierende Perspektive zwischen operativen und erläuternden Abschnitten, *um den Kommentar im Code zum eigentlichen Text zu machen*.

Das Verfahren der Quellcodekritik umfasst also zweierlei: auf einer pragmatischen Ebene erlaubt es, Softwarecode nicht einfach nur zur

38 Vgl. etwa Moritz Hiller: Diskurs/Signal (II). Prolegomena zu einer Philologie digitaler Quelltexte. In: Editio. Internationales Jahrbuch für Editionswissenschaft 28 (2014), S. 192–212; Thorsten Ries: »die geräte klüger als ihre besitzer«: Philologische Durchblicke hinter die Graphical User Interface. Überlegungen zur digitalen Quellenphilologie, mit einer textgenetischen Studie zu Michael Speiers *ausfahrt st. nazaire*. In: Editio. Internationales Jahrbuch für Editionswissenschaft 24 (2010), S. 149–199; Montfort: Exploratory programming; sowie die entsprechenden digitalen Erschließungsprojekte am Deutschen Literaturarchiv Marbach.

Anwendung zu bringen, sondern ihn systematisch und punktuell einer kritischen Lektüre zu unterziehen, die Algorithmen also ihrerseits lesbar und plausibel zu machen durch Erläuterungen, Reflexionen, Hinweise und gegebenenfalls auch durch ihre Modifikation. Dabei zielt diese auf Transparenz und Verständlichkeit ausgerichtete Methodik nicht allein auf Didaktik, insofern das Verstehen von Programmstrukturen als Stärkung einer *coding literacy* zu begreifen ist. Mehr noch, auf einer epistemischen Ebene, die wiederum weit über die – von Knuth (1984) und dem Prinzip des *Literate Programming* – intendierten Effekte hinausführt, geht es darum, mit Hilfe extensiven Kommentierens den Code zu narrativieren, zu historisieren und zu diskursivieren, mit einem Wort: den fortlaufenden Kommentar im Code zum eigentlichen Text zu erheben, was nichts anderes bedeutet, als den *Code immer schon als Geschichte zu schreiben*. Im Ergebnis heißt das, ein Modell der Quellenkritik des 21. Jahrhunderts zu entwickeln, um nichts weniger als einen neuen Standard des Schreibens von Code in den Geisteswissenschaften zu setzen.[39]

Damit birgt dieses Verständnis von Code nicht zuletzt ein emanzipatorisches Potenzial: Quellcodekritik bedeutet *coding literacy* auf einem Niveau, das die Machtverhältnisse in digitalen Gesellschaften zu reflektieren – und vielleicht sogar zu nivellieren – verhilft. Denn die Kulturtechnik Programmieren versetzt in die Lage, Code zu verstehen, einzuordnen und mithin auch zu schreiben, um angesichts der herrschenden Kräfte gegenwärtiger Informationstechnologien eine spezifische Handlungsmacht zurückzugewinnen.

Literaturverzeichnis

Alpaydın, Ethem: Machine learning. The new AI. MIT Press essential knowledge. Cambridge, MA 2016.

39 Mehr zu diesem breiter abgestützten Versuch bei Bajohr/Krajewski: Quellcodekritik.

Bajohr, Hannes: Algorithmic Empathy: On Two Paradigms of Generative Digital Literature. In: bmc:ct. Working Papers 1, Nr. 2 (2020), DOI: 10.12685/bmcct.2020.004.

Bajohr, Hannes/Markus Krajewski (Hg.): Quellcodekritik: Zur Philologie von Algorithmen. Berlin 2024.

Berry, David M./Anders Fagerjord: Digital Humanities. Knowledge and Critique in a Digital Age. Oxford 2017.

Brown, Tom u.a.: Language Models are Few-Shot Learners, 2020, arxiv.org.

Burnett, David Graham: Trying Leviathan. The nineteenth-century New York court case that put the whale on trial and challenged the order of nature, Princeton 2007.

Cox, Geoff/Christopher Alex McLean: Speaking code. Coding as aesthetic and political expression. Software studies. Cambridge, MA 2013.

Domingos, Pedro: The master algorithm. How the quest for the ultimate learning machine will remake our world. New York 2015.

Genette, Gérard: Paratexte. Das Buch vom Beiwerk des Buches [1992]. Frankfurt a.M. und New York 1997.

Grésillon, Almuth: Literarische Handschriften. Einführung in die critique génétique. Bern 1999.

Hilgers, Philipp von: Ursprünge der Black Box. In: Ofak, Ana/Philipp von Hilgers (Hg.): Rekursionen. Von Faltungen des Wissens. München 2010, S. 135–153.

Hiller, Moritz: Diskurs/Signal (II). Prolegomena zu einer Philologie digitaler Quelltexte. In: Editio. Internationales Jahrbuch für Editionswissenschaft 28 (2014), S. 192–212.

Kirschenbaum, Matthew: What Is Digital Humanities and What's It Doing in English Departments? In: ADE Bulletin 47, Nr. 150 (2010), S. 55–61.

Kirschenbaum, Matthew: Hello Worlds. Why humanities students should learn to program, 2009, https://mkirschenbaum.wordpress.com/2010/05/23/hello-worlds/(23.2.2024).

Knuth, Donald E.: Computer Programming as an Art. In: Communications of the ACM 17, Nr. 12 (1974), S. 667–673.

Knuth, Donald E.: Literate Programming. In: The Computer Journal 27 (1984), S. 97–111.

Krajewski, Markus: Against the Power of Algorithms. Closing, Literate Programming, and Source Code Critique. In: Law Text Culture 23 (2019), S. 119–133.

Krajewski, Markus: Hilfe für die digitale Hilfswissenschaft. Von den Digital Humanities verspricht man sich wahre Wunder, obwohl sie nur eine einfache Hilfswissenschaft sind. In: Frankfurter Allgemeine Zeitung, Nr. 85, N4, 2019.

Krajewski, Markus: branch, diff, merge. Versionskontrolle und Quellcodekritik. In: Jörg Paulus/Andrea Hübener/Fabian Winter (Hg.): Duplikat, Abschrift & Kopie. Kulturtechniken der Vervielfältigung. Wien und Köln 2020, S. 21–40.

Krajewski, Markus: Versionskontrolle. github.com/nachsommer. 2020.

Krajewski, Markus/Cornelia Vismann: Kommentar, Code und Kodifikation. In: Zeitschrift für Ideengeschichte 3, Heft 1 (2009), Themenheft »Kommentar«, S. 5–16.

Krämer, Sybille: Operative Schriften als Geistestechnik. Zur Vorgeschichte der Informatik. In: Peter Schefe u.a. (Hg.): Informatik und Philosophie. Mannheim und Leipzig 1993, S. 69–83.

Krysa, Joasia/Grzesiek Sedek: Source Code. In: Matthew Fuller (Hg.): Software studies. A lexicon. Cambridge, MA, und London 2008, S. 232–243.

Lanza, Emily: Who Painted Rembrandt? Copyright and Authorship of Two Rembrandt Portraits, 2018, https://www.thelegalpalette.com/home/2018/2/21/who-painted-rembrandt-copyright-and-authorship-of-two-rembrandt-portraits (nicht mehr online).

Lessig, Lawrence: Code. Version 2.0, 2. Aufl. New York 2006.

Manovich, Lev: Software takes command. New York 2013.

Mansfeld, Jaap (Hg.): Die Vorsokratiker. Griechisch/Deutsch, erw. Neuausg. Stuttgart 2012.

Marino, Mark C.: Critical Code Studies, 2006, https://electronicbookreview.com/essay/critical-code-studies/ (23.2.2024).

Marino, Mark C.: Critical Code Studies and the electronic book review: An Introduction, 2010, https://electronicbookreview.com/essay/cri

tical-code-studies-and-the-electronic-book-review-an-introductio n/ (23.2.2024).
Marino, Mark C.: Reading »exquisite_code«. In: N. Katherine Hayles/ Jessica Pressman (Hg.): Comparative textual media. Transforming the humanities in the postprint era. Minneapolis 2013, S. 283–309.
Marino, Mark C.: Critical code studies. Cambridge, MA 2020.
McCulloch, Warren: Recollections of the Many Sources of Cybernetics. In: ASC Forum 6, Nr. 2, 1974, S. 5–16.
Montfort, Nick: Exploratory programming for the arts and humanities. Cambridge, MA 2016.
Moretti, Franco: Distant reading. London 2013.
Pasquale, Frank: The Black box society. The secret algorithms that control money and information. Cambridge, MA 2016.
Passig, Kathrin: Fünfzig Jahre Black Box. In: Merkur. Deutsche Zeitschrift für europäisches Denken 71, Nr. 823 (2017), S. 16–30.
Punday, Daniel: Computing as Writing. Minneapolis 2015.
Pynchon, Thomas: Gravity's Rainbow. New York 1973/1974 (Erstveröffentlichung 1973).
Ramsay, Stephen: Reading machines. Toward an algorithmic criticism. Urbana 2011.
Ries, Thorsten: »die geräte klüger als ihre besitzer«: Philologische Durchblicke hinter die Graphical User Interfaces. Überlegungen zur digitalen Quellenphilologie, mit einer textgenetischen Studie zu Michael Speiers *ausfahrt st. nazaire*. In: Editio. Internationales Jahrbuch für Editionswissenschaft 24 (2010), S. 149–199.
Samek, Wojciech u.a.: Explainable AI: Interpreting, Explaining and Visualizing Deep Learning. Cham 2019.
Saxer, Daniela: Die Schärfung des Quellenblicks. Forschungspraktiken in der Geschichtswissenschaft 1840–1914. München 2014.
Schüttpelz, Erhard: Die medienanthropologische Kehre der Kulturtechniken. In: Archiv für Mediengeschichte 6 (2006), S. 87–110.
Siegert, Bernhard: Cultural Techniques: Or the End of the Intellectual Postwar Era in German Media Theory. In: Theory, Culture & Society 30, Nr. 6 (2013), S. 48–65.

Siegert, Bernhard: Cultural Techniques. Grids, Filters, Doors, and other Articulations of the Real. New York 2015.

Vee, Annette: Coding literacy. How computer programming is changing writing. Cambridge, MA 2017.

Vismann, Cornelia/Markus Krajewski: Computer-Juridisms. In: Grey Room. Architecture, Art, Media, Politics 8, Nr. 29 (2007), Special Issue: »New German Media Theory«, S. 90–109.

Winthrop-Young, Geoffrey: Friedrich Kittler zur Einführung. Hamburg 2005.

Winthrop-Young, Geoffrey: The Kultur of Cultural Techniques. Conceptual Inertia and the Parasitic Materialities of Ontologization. In: Cultural Politics 10, Nr. 3 (2014), S. 376–388.

Winthrop-Young, Geoffrey: Discourse, Media, Cultural Techniques: The Complexity of Kittler. In: Modern Language Notes 130, Nr. 3 (2015), S. 447–465.

Winthrop-Young, Geoffrey: Siren Recursions. In: Stephen Sale/Laura Salisbury (Hg.): Kittler Now: Current Perspectives in Kittler Studies. Cambridge 2015, S. 71–94.

Winthrop-Young, Geoffrey: The Kittler Effect. In: New German Critique 44, Nr. 132/3 (2017), S. 205–224.