text praxis

Digitales Journal für Philologie

Sonderausgabe # 8:

Generative Literatur.
Produktion und Rezeption im Zeichen des Codes

Hg. v. Stephanie Catani, Marlene Meuer und Niels Penke

Autor

Johannes Leitgeb (Würzburg)

Titel

Der Code des Autors. Transformationen generativer Literatur in Reimplementierung und Emulation

Erschienen in

Stephanie Catani, Marlene Meuer und Niels Penke (Hg.): *Generative Literatur. Produktion und Rezeption im Zeichen des Codes.* Sonderausgabe # 8 von *Textpraxis. Digitales Journal für Philologie* (1.2024)/www.textpraxis.net

URL: https://www.textpraxis.net/johannes-leitgeb-der-code-des-autors

URN: urn:nbn:de:hbz:6-86988447253

DOI: 10.17879/86988447019

URN und DOI dienen der langfristigen Auffindbarkeit des Dokuments.

Empfohlene Zitierweise

Johannes Leitgeb: »Der Code des Autors. Transformationen generativer Literatur in Reimplementierung und Emulation«. In: Stephanie Catani, Marlene Meuer und Niels Penke (Hg.): *Generative Literatur. Produktion und Rezeption im Zeichen des Codes.* Sonderausgabe # 8 von *Textpraxis. Digitales Journal für Philologie* (1.2024). URL: https://www.textpraxis.net/johannes-leitgeb-der-code-des-autors, DOI: 10.17879/86988447019.

Impressum

Textpraxis. Digitales Journal für Philologie
ISSN 2191-8236

Universität Münster
Graduate School Practices of Literature
Germanistisches Institut
Schlossplatz 34
48143 Münster

Redaktion dieser Ausgabe:
Nursan Celik, Dominik Hübschmann, Y. Su Kolsal, Rita Maricocchi, Michael A. Mason, Dominik Mika, Laura Reiling

textpraxis@uni-muenster.de

text praxis

Digital Journal for Philology

Special Issue # 8:

Generative Literatur.
Produktion und Rezeption im Zeichen des Codes

Ed. by Stephanie Catani, Marlene Meuer and Niels Penke

Author

Johannes Leitgeb (Würzburg)

Title

Der Code des Autors. Transformationen generativer Literatur in Reimplementierung und Emulation

Published in

Stephanie Catani, Marlene Meuer and Niels Penke (ed.): *Generative Literatur. Produktion und Rezeption im Zeichen des Codes.* Special Issue # 8 of *Textpraxis. Digital Journal for Philology* (1.2024)/www.textpraxis.net/en

URL: https://www.textpraxis.net/en/johannes-leitgeb-der-code-des-autors

URN: urn:nbn:de:hbz:6-86988447253

DOI: 10.17879/86988447019

URN und DOI serve the long-term searchability of the document.

Recommended citation

Johannes Leitgeb: »Der Code des Autors. Transformationen generativer Literatur in Reimplementierung und Emulation«. In: Stephanie Catani, Marlene Meuer and Niels Penke (ed.): *Generative Literatur. Produktion und Rezeption im Zeichen des Codes.* Special Issue # 8 of *Textpraxis. Digital Journal for Philology* (1.2024). URL: https://www.textpraxis.net/en/johannes-leitgeb-der-code-des-autors, DOI: 10.17879/86988447019.

Imprint

Textpraxis. Digital Journal for Philology
ISSN 2191-8236

Universität Münster
Graduate School Practices of Literature
Germanistisches Institut
Schlossplatz 34
48143 Münster
Germany

Editorial Board of this Issue:
Nursan Celik, Dominik Hübschmann, Y. Su Kolsal, Rita Maricocchi, Michael A. Mason, Dominik Mika, Laura Reiling

textpraxis@uni-muenster.de

Johannes Leitgeb (Würzburg)

Der Code des Autors

Transformationen generativer Literatur in Reimplementierung und Emulation

1 Einleitendes

1.1 »HONEY DEAR«

Betritt eine Besucherin des ZKM Karlsruhe im April 2009 die Installation des deutschen Künstlers David Link, findet sie sich in einem dunklen Kubus wieder.[1] An einer Wand hängen Glaszylinder, auf deren runden Enden leuchten grüne Punkte. Vor dieser Wand steht ein Bedienpult, das dutzende Schalter und Knöpfe beherbergt. Wenn sie mit diesem Bedienpult interagiert, ändern sich die Muster der grünen Punkte. Steht nun zeitgleich ein Besucher außerhalb dieses Kubus, kann er daraufhin auf einem Bildschirm lesen:

> HONEY DEAR
> YOU ARE MY SEDUCTIVE ARDOUR. MY LOVING LIKING SIGHS FOR YOUR FANCY. MY LONGING KEENLY HOPES FOR YOUR ENCHANTMENT. MY FERVENT CHARM LOVINGLY DESIRES YOUR PASSIONATE LONGING.
> YOU ARE MY DEVOTED HUNGER.
> YOURS ARDENTLY
> MUC[2]

David Link simuliert mit seiner Installation die Produktionsbedingungen eines der bekanntesten Werke generativer Literatur: Christopher Stracheys *Love Letter Algorithm*. Das beschriebene Bedienpult gleicht dem der *Ferranti Mark 1*, dem Nachfolger des Computers, den Strachey 1952 für die Ausgabe seiner Texte nutzte. Doch obwohl die Bedienung eng dem historischen Original folgt – so muss auch die Besucherin 2009 ihren Namen umständlich mit den Schaltern in Baudot-Code übersetzen – liegt ein neuralgischer Unterschied vor: Das Bedienpult ist nicht an einen raumgroßen Röhrenrechner angeschlossen, sondern an einen modernen PC, der den ursprünglichen Algorithmus Stracheys als Programm ablaufen lässt. Es handelt sich dabei um den Prozess der *Emulation* – der Nachahmung historischer Hardware auf einem modernen System. Emulation ermöglicht im Kontext der *software preservation*, digitale Objekte in ihrer vermeintlich

---

1 | Für eine Beschreibung der Installation vgl. David Link: »LoveLetters_1.0«, 2009, http://www.alpha60.de/art/love_letters/ (zuletzt eingesehen am 06. August 2023).

2 | Bei dem Zitat handelt es sich um einen der tatsächlich generierten Liebesbriefe: http://www.alpha60.de/art/love_letters/archive/muc/11-01-14_10-59.html (zuletzt eingesehen am 06. August 2023).

ursprünglichen Form zu archivieren, ohne die meist viel stärker dem materiellen Verfall ausgesetzten Hardware-Systeme lauffähig erhalten zu müssen.

Mit Verfahren der Emulation wird ein großes Nutzungspotenzial eröffnet, das nicht zuletzt für den Erhalt digitaler Literatur Potenziale bietet. Dennoch finden sich auf diesem Feld vor allem Versuche der vollständigen Neuprogrammierung der historischen Algorithmen in moderneren Programmiersprachen, bei denen der ursprüngliche Code gerade nicht erhalten bleibt. Eine geeignete Grundlage für die Analyse dieses Phänomens bieten die 1959 programmierten *Stochastischen Texte* von Theo Lutz, für die eine große Menge an Reimplementierungen vorliegt: im Rahmen von Kunstinstallationen, im Rahmen wissenschaftlicher Präservationsbestrebungen oder infolge privaten Interesses von Programmierer*innen (vgl. Tab. 1). Diese überaus heterogene Menge an Reimplementierungen als Rezeptionsleistungen zu lesen, zu erschließen und zu vergleichen, ist eines der Ziele dieses Aufsatzes. Grundlage ist die Rekonstruktion des Reinszenierungsprozesses als kommunikative Rezeption. Stefan Höltgen schreibt in seiner Monografie zur Entwicklung einer Methodik der ›Computerarchäologie‹: »Programmtexte haben häufig dazu eingeladen von Personen, die nicht ihre Autoren sind, modifiziert zu werden. Ein solcher Umschreibprozess stellt eine Rezeption im hier verstandenen Sinne dar.«[3] Meine darauf aufbauende These lautet, dass infolge dieses ›Umschreibprozesses‹ in der Reimplementierung generativer Literatur weder der ursprüngliche Code noch die originale Rezeptions- und Produktionssituation erhalten bleiben. Stattdessen entstehen neue technische und ästhetische Objekte, die ihrerseits auf die Produktionsbedingungen ihrer Gegenwart verweisen. Für die wissenschaftliche Erschließung kann dagegen – und hierin besteht das weitere Ziel des Aufsatzes – die Emulation großes Potenzial bieten, den Code als philologisches und epistemisches Dokument zu analysieren und ferner zu erhalten.

Argumentativ geht die vorliegende Untersuchung in drei Schritten vor: Ausgehend von einer Erschließung des von Lutz verfassten Codes werden *erstens* epistemische und ästhetische Schnittstellen ermittelt, die Charakteristika der ursprünglichen Textgenerierung ausweisen (Abschnitt 2). *Zweitens* werden ausgewählte Reimplementierungen auf ihre Beziehung zu Lutz' Code hin untersucht. Ziel in diesem Schritt ist es, die Ambivalenz zwischen der Präservation des historischen Materials einerseits und des produktiven und modernisierenden Umgangs andererseits kritisch zur reflektieren (Abschnitt 3). *Drittens* wird die Emulation des ursprünglichen Programms auf einem Emulator der Zuse Z 22 besprochen und daran anschließend die Bedeutung von Emulation für die Erschließung früher generativer Literatur diskutiert (Abschnitt 4).

Zunächst jedoch müssen in Abschnitt 1.2 einige theoretische Annahmen und Begriffe eingeführt werden, wie sie vor allem David Link und Stefan Höltgen im Rahmen der Definition einer Computerarchäologie verwenden. Ferner werden in Abschnitt 1.3 die Termini der ›Emulation‹, der ›Reimplementierung‹ und der ›Simulation‹ voneinander abgegrenzt.

1.2 Methodik

Die zentrale methodische Herausforderung bei der Betrachtung (historischer) digitaler Literatur ist der Umstand, dass Computer und digitale Maschinen sich als Medium stets in einem *Prozess* befinden: »Computer sind Medien, die aufgrund ihrer Operationen *speichern, übertragen und prozessieren* in der Lage sind alle anderen Medien zu

3 | Stefan Höltgen: *Open History. Archäologie des Retrocomputings*. Berlin 2022, S. 136.

simulieren. Um diese Eigenschaft zu erlangen, müssen Computer sich aber *im Vollzug* befinden, das heißt *operativ sein*.«[4] Diese Eigenschaft überträgt sich auch auf die Objekte und Artefakte, die in der Form von Dateien, Programmen und anderen digitalen Formaten als Ein- bzw. Ausgabe des Computers fungieren: Auch für diese gilt, dass sie ihre volle strukturelle Komplexität nur dann entfalten können, wenn der Computer, durch den sie vermittelt werden, ›operativ‹ ist.[5] Diese Eigenschaft erscheint umso weniger trivial, je größer die zeitliche Lücke zwischen dem Ursprung des Artefakts und der Analysesituation ist, stehen doch oftmals keine funktionsfähigen Exemplare älterer Computersysteme mehr zur Verfügung. Um diesen Faktoren in meiner Analyse Rechnung zu tragen, werde ich auf verschiedene methodische Frameworks zurückgreifen, insbesondere auf die Arbeiten Stefan Höltgens zur ›Computerarchäologie‹ sowie auf die von David Link entwickelte ›Archaeology of Algorithmic Artefacts‹. Charakteristisch für beide ist das Primat eines Hands-on-Ansatzes: Aufgrund der oben angesprochenen Eigenschaften digitaler Medien ist es das erklärte Ziel der Computerarchäologie, die Analyseergebnisse auf einem operationalen Computer zu demonstrieren, da »nur durch die *Demonstration* der Argumente am Objekt einer rein diskursiven Formulierung [...] entkommen werden kann«.[6] An dieser Stelle können Emulatoren in die Arbeitsweise eingegliedert werden: Denn ist diese operationale Form auf dem originalen Gerät nicht herzustellen, so bietet die Emulation eine geeignete Heuristik.[7] Auch Methoden der Code- und Computerphilologie[8] werden im vorliegenden Beitrag als Werkzeug der Computerarchäologie verwendet, lassen sich doch zumeist bereits auf Ebene des Programmcodes epistemische Schnittstellen erkennen.

Auf methodologischer Ebene lässt sich schließlich der Anschluss an mehrere einschlägige Medien- und Kulturtheorien suchen: Zunächst muss mit Friedrich Kittler argumentiert werden, dass eine unmittelbare Abhängigkeit der Software von der Hardware besteht. Dieses von ihm eingeforderte medientheoretische Apriori der Hardware[9] erfordert die Relationierung einer Ebene digitaler Prozesse einerseits und einer Ebene physischer Manipulationen eines Mediums andererseits. Diese Beziehung ist ferner die Grundlage, um die im vorliegenden Beitrag betrachteten Artefakte als unikale Konstellationen von Maschine und Code zu identifizieren. Eine zweite medientheoretische Schnittstelle ergibt sich mit der Perspektivierung der Methodik als ›Archäologie‹[10] im Sinne Foucaults. Dabei gilt für die Methode der Computerarchäologie ebenso wie die von Foucault propagierte: »The archaeology of algorithmic artefacts endeavours to

---

4 | Ebd., S. 13.

5 | Vgl. David Link: *Archaeology of Algorithmic Artefacts*. Minneapolis 2016, S. 100.

6 | Höltgen: *Open History*, S. 80.

7 | Vgl. Simon Dor: »Emulation«. In: Mark J. P. Wolf u. Bernard Perron (Hg.): *The Routledge Companion to Video Game Studies*. New York 2013, S. 25–31, hier S. 26. Vgl. auch Höltgen: *Open History*, S. 141.

8 | Vgl. Höltgen: *Open History*, S. 129.

9 | Vgl. Friedrich Kittler: »Es gibt keine Software«. In: Ders.: *Draculas Vermächtnis. Technische Schriften*. Leipzig 1993, S. 225–242, hier S. 237.

10 | Der zitierte Archäologie-Begriff bei Foucault folgt hier dem in der *Archäologie des Wissens*: »Die archäologische Analyse individualisiert und beschreibt diskursive Formationen. Das heißt, sie muß sie in der Gleichzeitigkeit, in der sie sich präsentieren, konfrontieren und sie einander gegenüberstellen, sie von denen unterscheiden, die nicht dieselbe Zeitrechnung haben, sie in ihrer Spezifität mit den nicht diskursiven Praktiken in Beziehung setzen, die sie umgeben und ihnen als allgemeines Element dienen« (Michel Foucault: *Archäologie des Wissens*. Übers. v. Ulrich Köppen. Frankfurt a. M. 1981, S. 224).

reconstruct from objective technical forms the theoretical currents that generated them and were generated by them.«[11] Diese Rekonstruktionsleistung kann nur gelingen, indem das Artefakt, das – in der Sprache von Höltgen bildlich bezeichnet – »von den Diskursen, Historiographien, Anekdoten und Mythen verdeckt ist«,[12] von diesen losgelöst wird. Damit ist ein weiterer Schritt in der Methode angesprochen: Um Lutz' originalen Code zu erschließen, müssen sowohl der Code selbst philologisch betrachtet als auch die umliegenden Diskurse der Informatik berücksichtigt werden.

1.3 Emulation und Reimplementierung

Einige der im vorliegenden Beitrag verwendeten Termini bedürfen vorab einer begrifflichen Klärung – nicht zuletzt der Nexus um die Begriffe der ›Emulation‹ und der ›Reimplementierung‹ eines Computerprogramms. Der Begriff eines Emulators ist im *Retrocomputing* grundlegend im folgenden Sinn zu verstehen: »An emulator is an application that tries to mimic another system in order to run applications the way they were run on their original system.«[13] Medientheoretisch abstrahiert lässt sich dabei ein ›Zeichenwechsel‹ feststellen: »Grundsätzlich übersetzen Software-Emulatoren physikalische in semiotische Zeichenkategorien: Ihr Ziel ist es die *Hardwarestruktur* des Originalsystems in einer *Software* abzubilden, die auf dem Hostsystem ausgeführt wird.«[14] Grundprinzip der Emulation ist es, dass »gleiche Eingaben zu gleichen Ausgaben führen«.[15]

Bedingung der Emulation ist die hierarchische Gliederung digitaler Objekte in physische, logische und konzeptuelle Bestandteile nach Thibodeau: »A *physical* object is simply an inscription of signs on some physical medium. A *logical* object is an object that is recognized and processed by software. The *conceptual* object is the object as it is recognized and understood by a person [...].«[16] Damit lassen sich digitale Objekte auf verschiedenen Ebenen erhalten, erschließen und analysieren: Das Objekt wird auf physischer Ebene zur Entität, die ohne das ursprüngliche System Bestand haben kann, die logische Ebene kann durch den Einsatz eines Emulators konstruiert werden und nur die (originale) konzeptuelle Ebene kann durch das Originalsystem erzeugt werden. Sogenannte *Skeumorphismen* – Elemente des Originalsystems, die in der Emulation ohne Zweck auftreten und oftmals lediglich »schmückenden Charakter«[17] annehmen – versuchen diese Distanz zu überbrücken und auf ästhetischer Ebene das Originalsystem nachzubilden. Demgegenüber steht die ›Reimplementierung‹, die im vorliegenden Beitrag die (Re-)Programmierung eines Algorithmus in einer anderen Programmiersprache

---

11 | Link: *Archaeology of Algorithmic Artefacts*, S. 111.

12 | Höltgen: *Open History*, S. 71.

13 | Dor: »Emulation«, S. 25.

14 | Höltgen: *Open* History, S. 247.

15 | Claus Pias: »Medienphilologie und ihre Grenzen«. In: Friedrich Balke u. Rupert Gaderer (Hg.): *Medienphilologie. Konturen eines Paradigmas*. Göttingen 2017, S. 365–385, hier S. 377.

16 | Kenneth Thibodeau: »Overview of Technological Approaches to Digital Preservation«. In: Council on Library and Information Resources (Hg.): *The State of Digital Preservation: An International Perspective*. Washington, D. C. 2002, S. 4–31, hier S. 6.

17 | Jens-Martin Loebel: *Lost in Translation. Leistungsfähigkeit, Einsatz und Grenzen von Emulatoren bei der Langzeitbewahrung digitaler multimedialer Objekte am Beispiel von Computerspielen*. Berlin 2013, S. 127.

bezeichnet. Dabei kommt es zu starken Migrations-Effekten,[18] die das digitale Objekt weder auf physischer noch auf logischer oder konzeptueller Ebene erhalten.[19]

2 »T1700T«: Lutz' Stochastische Texte in ihrer Zeit

Lutz programmiert die *Stochastischen Texte* 1959 auf der Zuse Z 22 der Technischen Hochschule Stuttgart.[20] Die Bedienung der Z 22 erfolgt über ein Bedienpult, das über mehrere Befehlstasten und Leuchten zur Anzeige des aktuellen Status verfügt. Die Ein- und Ausgabe von Daten und Programmen geschieht über das analoge Medium Papier: entweder in der Form von Lochstreifen oder als Fernschreiberausdruck. Eine der möglichen Programmiersprachen, die auch von Lutz für die Programmierung der *Stochastischen Texte* verwendet wurde, ist der sogenannte Freiburger Code,[21] eine assemblernahe Sprache, die vor allem über arithmetische und logische Befehle verfügt. Der innere Zustand der Maschine und etwaige Fehler im Durchlauf des Programms bleiben über diese Möglichkeiten der Ausgabe hinaus verborgen. Dadurch entsteht ein besonderes Verhältnis der frühen Computer zur Materialität, das eine erste Schnittstelle für die Analyse birgt: Code ist das Zusammenspiel analoger und digitaler Komponenten und Maschinen. Der Erhalt des Codes als physisches Objekt nach Thibodeau ist im Fall von Lutz' *Stochastischen Texten* auf dem gänzlich analogen Speicherträger Papier gewährleistet: Im Deutschen Literaturarchiv (DLA) Marbach liegen ein Fernschreiberausdruck des Programms für die Generierung der Texte sowie die unredigierte Fassung der ausgegebenen Textfragmente.[22] Auf diesen Ausdruck, auf Lutz' Artikel sowie auf zeitgenössische Quellen zur Z 22 stützt sich die nachfolgende Analyse, um an zwei zentralen Passagen im Code von Lutz seine Herangehensweise in a) der algorithmischen und b) in der technischen Lösung des Problems stochastischer Texte aufzuzeigen. Ziel ist es dabei, Charakteristika von Lutz' Werk zu ermitteln, die im nachfolgenden Kapitel den Reinszenierungen gegenübergestellt werden können.

---

18 | Vgl. Thibodeau: »Overview of Technological Approaches to Digital Preservation«, S. 18f.

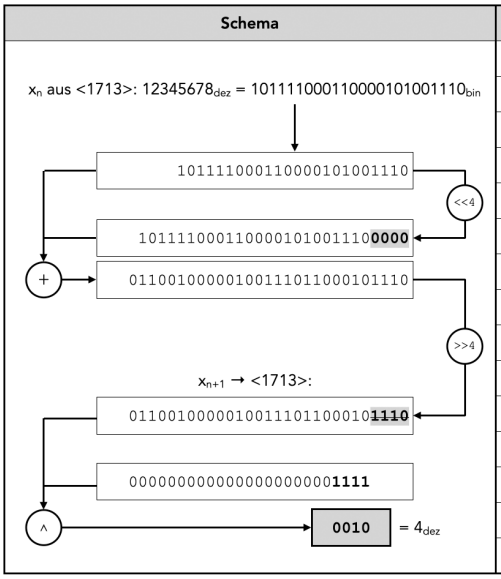
19 | Dass auch Emulation Migrationsstrategien beinhaltet, da Emulatoren in einer Programmiersprache kodiert sind, muss im Rahmen der Analyse in den Hintergrund treten (vgl. ebd., S. 20).

20 | Vgl. Toni Bernhart: »Beiwerk als Werk. Stochastische Texte von Theo Lutz«. In: *editio* 34.1 (2020), S. 180–206, hier S. 195, DOI: 10.1515/editio-2020-0010,. Zur institutionellen Verankerung der Stuttgarter Z 22 vgl. Christoph Hoffmann: »Eine Maschine und ihr Betrieb. Zur Gründung des Recheninstituts der Technischen Hochschule Stuttgart (1956–1964)«. In: Ders., Barbara Büscher u. Hans-Christian von Herrmann (Hg.): *Ästhetik als Programm. Max Bense/Daten und Streuungen*. Berlin 2004, S. 119–129.

21 | Vgl. Bernhart: »Beiwerk als Werk«, S. 195.

22 | Ein Faksimile des generierten Textes findet sich in Theo Lutz: »Stochastische Texte« [1959]. In: Barbara Büscher, Christoph Hoffmann u. Hans-Christian von Herrmann (Hg.): *Ästhetik als Programm. Max Bense/Daten und Streuungen*. Berlin 2004, S. 164–169, hier S. 169. Das Programm ist bisher nicht publiziert; die Bereitstellung einer zeichengenauen Abschrift auf GitHub für die Verwendung in Emulatoren ist seitens des Autors dieses Beitrags in Vorbereitung.

2.1 Die Stochastischen Texte in der algorithmischen Umsetzung



#	Code	Erklärung
	T1700T	Bandbefehl: Das nachfolgende Programm startet ab Trommelspeicheradresse <1700>.
1700	B5	Bringe den Inhalt aus Speicherzelle <5> in den Akkumulator (*a*).
1701	T1712	Speichere den Inhalt von *a* in Speicherzelle <1712>.
1702	B1713	Bringe den Inhalt aus Speicherzelle <1713> in *a*.
1703	LLA0	Verschiebe den Inhalt von *a* um 2 Bits nach links (2x); bewirkt bei Ganzzahlen eine Multiplikation mit 4.
1704	LLA0	
1705	A1713	Addiere den Inhalt von <1713> und den Inhalt von *a*.
1706	RA0	Verschiebe den Inhalt von *a* um 1 Bit nach rechts (4x); bewirkt bei Ganzzahlen eine abgerundete Division durch 2.
1707	RA0	
1708	RA0	
1709	RA0	
1710	U1713	Speichere den Inhalt von *a* in Speicherzelle <1713> um.
1711	CI15	Bilde die Intersektion aus dem Inhalt von *a* und der Zahl 15 (= 1111_{bin}).
1712	0	*Leerzelle* (diese Speicherzelle wird überspeichert, s. <1701>).
1713	12345678'	Ganzzahl 12345678 als Seed (diese Speicherzelle wird überspeichert, s. <1710>).

Abb. 1: Schematische Aufschlüsselung einer einzelnen Iteration von Lutz' Zufallszahlengenerator.

Eingangs kann auf die Ebene des Algorithmus eingegangen werden. Eine erste neuralgische Stelle findet sich bereits in den ersten Zeilen des Programms: Lutz' Implementierung eines Zufallszahlengenerators. Beginnend mit dem Bandbefehl ›T1700T‹[23] wird eine Funktion entworfen, die als Eingabe eine beliebige Zahl als Ausgangspunkt (d. i. als *Seed*) benötigt und am Ende eine (pseudo-)zufällige vierstellige Binärzahl – also eine Zahl im Intervall von 0 bis 15 – ausgibt. Diese Zahl wird im weiteren Programmverlauf verwendet, um auf die in 16 aufeinanderfolgenden Trommelspeicherzellen abgelegten Einträge des Wörterbuchs zuzugreifen. Diese Funktionsweise des Zufallsgenerators legt Lutz auch schematisch in dem die *Stochastischen Texte* begleitenden Artikel dar.[24] Von Interesse ist dabei, dass Lutz die Erzeugung von Zufallszahlen bereits um 1960 als ein gelöstes Problem ansieht: »Mit der Existenz eines solchen Zufallsgenerators ist das Problem der stochastischen Texte im wesentlichen gelöst.«[25] Diese Aussage kann durch einen Blick auf zeitgenössische Episteme in der Informatik perspektiviert werden. Obwohl das Problem randomisierter Zahlenreihen bereits in der Mathematik des 19. Jahrhunderts Aufmerksamkeit erlangt, sind algorithmische Ansätze um 1960 noch recht jung.[26] Wurden zuvor analoge Geräte oder umfassende Tabellenwerke als Grundlage für Zufallszahlen verwendet, entstehen erst um 1946 Versuche, Reihen an Zufallszahlen in einzelnen Schritten mathematisch – bzw. algorithmisch – zu erzeugen.[27] Gemeinsamkeit dieser Ansätze ist, dass es sich bei den ausgegebenen Zahlen nicht um wirklich zufällige Reihen

23 | Theo Lutz: [Fernschreiberausdruck mit der handschriftlichen Angabe »Programm für STOCHASTISCHE TEXTE. 2. Auflage. 29. Juli 1959«]. Deutsches Literaturarchiv Marbach (DLA, A: Lutz, Theo, Kasten 2, Mappe 3) (zitiert als STP), o. S.

24 | Vgl. Lutz: »Stochastische Texte«, S. 165f.

25 | Ebd., S. 166.

26 | Vgl. Pierre L'Ecuyer: »History of Uniform Random Number Generation«. In: W. K. V. Chan u. a. (Hg.): *Proceedings of the 2017 Winter Simulation Conference*, Las Vegas 2017, S. 202–230, hier S. 202, DOI: 10.1109/WSC.2017.8247790.

27 | Vgl. ebd., S. 205.

handelt, sondern um *Pseudo-Zufallszahlen*, deterministische Zahlenreihen, die eine möglichst hohe Periode aufweisen, bevor sie sich wiederholen.[28]

Den Zufallsgenerator von Lutz im Spektrum dieser zeitgenössischen Algorithmen zu verorten, erlaubt zwei Rückschlüsse: *Erstens* ist diese Analyse zentral für den Vergleich der epistemischen und ästhetischen Eigenleistung Lutz' mit deren Reinszenierungen, da – wie gezeigt werden soll – die Ergebnisse des von Lutz implementierten Zufallszahlengenerators spezifische Auswirkungen auf die Ausgabe der Texte haben. *Zweitens* kann die Analyse exemplarisch die Wirkungsweisen codephilologischer Arbeit und deren Potenzial für eine literaturwissenschaftliche Analyse digitaler Texte aufzeigen.

Zunächst muss die Funktionsweise von Lutz' Zufallsgenerator in gebotener Kürze erschlossen werden. Der Algorithmus lässt sich schrittweise und schematisch aufschlüsseln (vgl. Abb. 1).[29] Auf dieser abstrahierten Ebene lassen sich Schnittstellen zu anderen zeitgenössischen Algorithmen finden, insbesondere zu den 1951/1958 entwickelten *Linear-congruential-generator*-Algorithmen (LCG) sowie den 1965 publizierten *Tausworthe*- bzw. *Linear-Feedback-Register*-Modellen (LFR). Besonders die LCG stellen bis heute eine einflussreiche Methode für die Generation von Pseudozufallszahlen dar.[30] Charakteristisch sind hier vor allem die lineare Kombination des *Seeds* sowie die nachfolgende *modulo*-Operation, um einen festen Wertebereich zu erreichen. Diese mathematischen Charakteristika weist auch der Code von Lutz auf, jedoch mit einer entscheidenden Abweichung: Anstatt der Multiplikation mit einer Konstanten, greift er auf *Bitshift*-Operationen zurück, die fünf Jahre später auch für die neuentwickelten LFR wesentlich werden. Grund dafür ist, dass die Z 22 nicht über einen expliziten Multiplikationsbefehl verfügt, sondern nur über *Bitshift*-Befehle, die Multiplikation mit Potenzen von 2 erlauben. Insgesamt lässt sich konstatieren, dass Lutz keine standardmäßige Methode für die Zufallszahlengenerierung wählt, da er verschiedene Elemente aus zeitgenössischen Theorien rekombiniert. Die deutlichen Übereinstimmungen sind Indizien dafür, dass Lutz als Mathematikstudent in Kontakt mit den zeitgenössischen Algorithmen für die Berechnung von Pseudozufallszahlen gekommen sein muss. Die Ermittlung weiterer konkreter Code- oder Literaturvorlagen ist an dieser Stelle zwar als Desiderat anzumerken; nichtsdestotrotz ist davon auszugehen, dass es sich bei der Programmierung des Zufallsgenerators um eine Eigenleistung Lutz' handelt, der in der Kombination verschiedener Algorithmen mit der Technik der Z 22 einen für die *Stochastischen Texte* spezifischen Zufallszahlengenerator programmiert hat.

28 | Vgl. ebd., S. 207.

29 | Nils Reiter formalisiert den Zufallsgenerator algorithmisch: , wobei die ausgegebene Zufallszahl und eine Modifikation des *Seed*-Wertes (Start-Wertes) ist (vgl. Nils Reiter: »Sitzung 7. Zufall« [Lehrmaterialien für die Übung Reenactment historischer Rechner], 2022, https://lehre.idh.uni-koeln.de/site/assets/files/3972/session-07.pdf (zuletzt eingesehen am 06. August 2023), S. 10).

30 | Vgl. L'Ecuyer: »History of Uniform Random Number Generation«, S. 207.

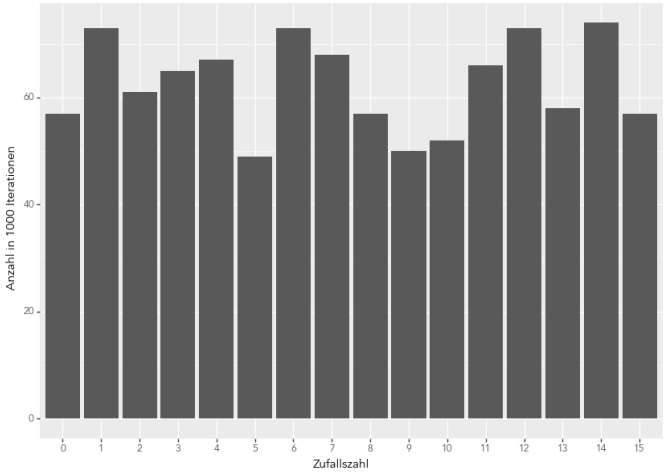


Abb. 2: Auswertung von Lutz' Zufallszahlengenerator nach 1000 Iterationen.

Der Blick auf zeitgenössische Algorithmen zeigt deutlich auf, dass starke Beschränkungen für die Erzeugung von Zufallszahlen vorliegen. Wenngleich sich mit den zeitgenössischen Zufallsgeneratoren eine hohe Periode erreichen lässt, handelt es sich wie oben erwähnt in jedem Fall um deterministische Zahlenreihen. Daraus ergeben sich zwei Folgen: *Einerseits* ist die Zahlenreihe bei Verwendung desselben *Seeds* reproduzierbar – auch zum gegenwärtigen Zeitpunkt lässt sich die Ausgabe von Lutz' Originaltext reproduzieren (Abschnitt 4 dieses Beitrags). *Andererseits* ist keine Gleichverteilung der Zahlenreihen gewährleistet: Deren Approximation ist wie die deterministische Zahlenreihe abhängig vom Ausgangswert. Für den von Lutz gewählten *Seed*[31] 12345678 stellt sich so ein sehr unausgewogenes Bild dar (vgl. Abb. 2). Die Wahl des *Seeds* lässt sich iterativ optimieren; die dafür notwendige Rechenleistung steht Lutz jedoch nicht zur Verfügung. Dass Lutz diese eher approximative Gleichverteilung aufgefallen ist, zeigen seine redaktionellen Eingriffe in die ausgegebenen Texte, wie auch Bernhart rekonstruiert: »Das technische Kalkül zielt darauf ab zu kaschieren, dass beim Programmlauf die zwei Substantive ›TISCH‹ und ›KNECHT‹ im generierten Text unberücksichtigt blieben. Die Gründe dafür sind vermutlich mathematischer Natur und in dem nicht zuverlässig operierenden Zufallsgenerator zu vermuten.«[32] Diese manuellen Korrekturen zeigen auf, wie zentral die heuristische Funktionsweise des Zufallsgenerators für die ausgegebenen Texte sowie für die publizierten Texte und damit für die Anlage des Werkes nach Lutz' mathematischer und ästhetischer Vorstellung ist. Nur Codephilologie – kritische Lektüre des Codes – und Hands-on-Ansatz – Durchlauf des Programms im Emulator – können diese Tiefenstrukturen aufdecken, da Lutz in seinem Aufsatz weder die Einschränkungen dokumentiert noch seine Implementierung kommentiert: Der Code des Autors bleibt für die zeitgenössischen Rezipient*innen unzugänglich.

2.2 Die Stochastischen Texte in der technischen Umsetzung

Eine zweite neuralgische Stelle findet sich in der Umsetzung des Lexikons. Dabei lässt sich – Kittlers Apriori der Hardware entsprechend – insbesondere auf die Limitationen und Bedingungen eingehen, die die Technik der Zuse Z 22 der Implementierung auferlegt.

31 | Vgl. STP.

32 | Bernhart: »Beiwerk als Werk«, S. 193.

Lutz gibt in seinem Aufsatz eine Liste an 16 Substantiven und 16 Adjektiven an, die als Lexikon für die Generierung der Texte dienen sollen.[33] Ein Blick auf den Code offenbart auf Ebene der Programmierung eine leicht abweichende Liste:

> GRAF, FREMDE, BLICK, KIRCHE, SCHLOS, BILD, AUGE, DORF, TURM, BAUER, WEG, GAST, TAG, HAUS, TISCH, KNECHR
>
> OFFEN, STILL, STARK, GUT, SCHMAL, NAH, NEU, LEISE, FERN, TIEF, SPAET, DUNKEL, FREI, GROSS, ALT, WUTEND[34]

Auffällig sind dabei die unterschiedlichen Schreibungen der Umlaute in *SPAET* und *WUTEND*, des *ß* in *SCHLOS* und *GROSS* sowie die falsche Schreibung in *KNECHR*. Ferner fallen in der Ausgabe des Textes unangepasste indefinite Formen bei *FREMDE* auf: »Ein Fremde ist leise«.[35] Weiterhin erscheinen in den ausgegebenen Texten Wörter, die in dieser Form nicht in der Wortliste zu finden sind, so z. B. *JEEER* und *HAS*. Mögliche Erklärungen dieser Unterschiede zum publizierten Text, die die spezifische Technik der Z 22 berücksichtigen, können im Folgenden dargestellt werden.

Zunächst ist es von Relevanz, die Speicherung von Wörtern in der Z 22 zu betrachten. Grundsätzlich gilt, dass in jeder Speicherzelle mit einer Größe von 38 Bits sieben »Buchstaben, Ziffern, Zeichen sowie Zeilentransport, Zwischenraum, Ziffern-Buchstaben-Umschaltung usw. des Fernschreibalphabetes«[36] abgespeichert werden können. Die Bits verteilen sich dabei wie folgt: Nach den drei Bits, die den Datentyp deklarieren, folgen fünf Bits für jedes Zeichen (vgl. Abb. 3).

A		B	Zeichen 1					Zeichen 2					Zeichen 3					Zeichen 4					Zeichen 5					Zeichen 6					Zeichen 7				
			S					*C*					*H*					*L*					*O*					*S*					_				
0	1	1	1	0	1	0	0	0	1	1	1	0	0	0	1	0	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0	0	0	0	1	0	0

A: 01 = Kennzeichen für Klartext
B: 1 = Fernschreiber auf »Buchstaben« stellen

Abb. 3: Binäre Repräsentation des Wortes ›Schlos ‹ als Maschinenwort mit 38-bit; Übersetzung in Binärcode nach dem International Telegraph Alphabet 2.

Deren Kodierung folgt dem Standard des *International Telegraph Alphabet 2* bzw. *CCITT-2* von 1931.[37] Damit ist die maximale Länge der Zeichenkette bei sieben Zeichen festgelegt. Für die Implementierung bei Lutz kommt jedoch noch eine zusätzliche Beschränkung hinzu: Um die Leerzeichen zwischen den Wörtern im ausgegebenen Text zu berücksichtigen, speichert er jedes Wort mit einem Spatium am Ende ab.[38] Die nutzbare Zeichenmenge reduziert sich so auf sechs Zeichen. Für die Auswahl der Wörter in den *Stochastischen Texten* kommen für Lutz daher grundsätzlich nur Formen infrage, die nicht länger als sechs Zeichen sind. Dieser Umstand kann eine Erklärung für die Schreibungen *WUTEND*

---

33 | Vgl. Lutz: »Stochastische Texte«, S. 166.

34 | Vgl. STP.

35 | Theo Lutz: [Fernschreiberausdruck ohne Titel; Incipit: »NICHT JEDER BLICK IST NAH«]. Deutsches Literaturarchiv Marbach (DLA, A: Lutz, Theo, Kasten 2, Mappe 3) (zitiert als STT), o. S.

36 | Zuse KG: *Programmgesteuerte Elektronische Rechenanlage Zuse Z 22 und Z 22 R. Programmierungsanleitung*. Bad Hersfeld 1960, S. 13.

37 | Vgl. Wolfgang Pavel: »Zuse Z22. Dokumentation und Simulation«, 23. Dezember 2013, https://www.wpavel.de/zuse/doku/doku_simu.php (zuletzt eingesehen am 06. August 2023).

38 | Die Kodierung mit einem Spatium lässt sich nicht definitiv belegen, da dieses auf dem Fernschreiberausdruck nicht unmittelbar ersichtlich ist. Im Code finden sich jedoch keine Hinweise auf eine anderweitige Implementierung.

sowie *SCHLOS* bieten, die in der ausgeschriebenen Form über das Zeichenlimit hinausgehen würden. Bei *FREMDE* liegt darüber hinaus ein weiteres Problem vor: Einerseits erreicht die Form bereits das Zeichenlimit, andererseits handelt es sich um das einzige Substantiv in der Menge der ausgewählten Wörter, das seine Form bei einem definiten Artikel ändert. Dieser Sonderfall wurde von Lutz nicht programmiert,[39] sodass die grammatikalisch falsche Form »Ein Fremde ist leise«[40] in der Ausgabe des Programms erscheint. Für die Publikation musste Lutz diese Formen anschließend bereinigen und somit die ursprüngliche Ausgabe des Programms redaktionell verändern.[41]

Die Erklärung der Form *KNECHR* sowie der nicht in der Wortliste erscheinenden Wörter *HAS* und *JEEER* ist weniger eindeutig. Klemens Krause vom Computermuseum der Stuttgarter Informatik geht bei *KNECHR* von einem simplen Schreibfehler aus.[42] Nils Reiter schlägt für die anderen Formen das Eintreten von ›Bitflips‹ vor, natürliche Phänomene, die einzelne Bits auf einem Speicherträger dazu veranlassen, von 0 auf 1 zu wechseln oder vice versa.[43] Besonders im Fall von *JEEER* wird diese Hypothese plausibel: Die *CCITT*-2-Kodierungen für *E* (10000) und für *D* (10010) unterscheiden sich lediglich um ein Bit. Auch tritt dieser Fehler nicht systematisch im Text auf. Eine abschließende Verifikation dieser Hypothese erfordert jedoch weitere Analysen, die im Rahmen dieses Beitrags nicht geleistet werden können.[44]

Zusammenfassend lässt sich festhalten, dass die *Stochastischen Texte* auf der Ebene sowohl ihres Programms als auch der Hardware der Z 22 charakteristische Limitationen aufweisen, die Einfluss auf die ausgegebenen Texte haben. Lutz hat beide Fehlerquellen wahrgenommen und redaktionelle Veränderungen am Text vorgenommen, sodass der publizierte Text allein nicht die unmittelbare Ausgabe der Maschine wiedergibt. Dennoch ist das ursprüngliche Programm als eigenes ästhetisches und epistemisches Objekt zu werten, das in den spezifischen produzierten Texten den eigenen Aufbau und die eigene Technik dokumentiert und damit auf die zeitgenössischen Episteme der Informatik um 1960 verweist.

---

39 | Vgl. Bernhart: »Beiwerk als Werk«, S. 200.

40 | STT.

41 | Vgl. Bernhart: »Beiwerk als Werk«, S. 192f.

42 | Vgl. Computermuseum der Stuttgarter Informatik: »Stream vom 22.6.2022 - Thema: Theo Lutz - Stochastische Texte«, *YouTube*, 11. Juli 2022, https://www.youtube.com/watch?v=9EfzNPxs82M (zuletzt eingesehen am 06. August 2023), 32:20.

43 | Vgl. Nils Reiter: »Sitzung 1. Einführung, Ablauf des Kurses, Organisatorisches« [Lehrmaterialien für die Übung Reenactment historischer Rechner], 2022, https://lehre.idh.uni-koeln.de/site/assets/files/3972/session-01.pdf (zuletzt eingesehen am 06. August 2023), S. 15.

44 | Anders verhält es sich mit *HAS*. Auch hier geht Krause von einem Schreibfehler aus, der jedoch infolge einer programmiertechnischen Besonderheit entsteht: Um die von Max Bense für die *Stochastischen Texte* geforderten 16 Substantive abzudecken, aber gleichzeitig den bereits entwickelten Zufallsgenerator beizubehalten, könne Lutz für eine frühere Version eine andere Wortliste mit nur 14 Substantiven benutzt haben. In dieser müssen – so Krause – einzelne Wörter doppelt besetzt gewesen sein. Nur bei der Verdopplung der Form *HAUS* sei daher der Schreibfehler passiert, weswegen die richtige neben der falschen Form im ausgegebenen Text erscheint (vgl. Computermuseum der Stuttgarter Informatik: »Stream vom 22.6.2022«, 32:55–34:06).

3 »from random import choice«: Lutz' *Stochastische Texte* heute

Wie eingangs beschrieben, existiert für Lutz' Code eine Vielzahl von Rezeptionszeugnissen, die den ursprünglichen Algorithmus aufgreifen, modifizieren und in eine andere Programmiersprache übersetzen. Einen Überblick über einige der Reimplementierungen kann die nachfolgende Darstellung bieten (vgl. Tab. 1; s. auch das Literaturverzeichnis).

Autor*in	Jahr	Programmiersprache / -plattform
Johannes Auer	2005	PHP
eddeaddad	2010	JanusNode
Nick Montfort	2014	JavaScript; Python
Toni Bernhart, Christian Corti, Klemens Krause (Computermuseum der Stuttgarter Informatik)	2022[45]	PDP-8-Assembler; LGP-30; PDP-12
Toni Bernhart, Nils Reiter	in Vorbereitung	Reinszenierung des Codes auf lauffähiger Z 22; Emulation

Tab. 1: Chronologische Übersicht ausgewählter Reimplementierungen von Lutz' Algorithmus der Stochastischen Texte.

Ausgehend von den im vorangegangenen Kapitel erarbeiteten Schnittstellen können diese Reimplementierungen auf den Umgang mit der Materialität, den Umgang mit Randomisierung sowie auf den Umgang mit technischen Limitationen der Z 22 hin analysiert werden. Eine weitere Gegenüberstellung wird das vierte Kapitel des vorliegenden Beitrags liefern, in dem der Durchlauf des originalen Codes auf einem Emulator der Z 22 reflektiert wird.

Grundsätzlich handelt es sich bei allen betrachteten Reimplementierungen um Neuprogrammierungen des ursprünglichen Algorithmus in anderen Programmiersprachen. Dieser Prozess versteht sich aus der Perspektive der Computerarchäologie als »Umschreibprozess«[46] oder als »Übersetzung«[47]. Diese Begriffe können im Folgenden synonym verwendet werden, da sich beide Begriffe von der ›echten‹ Emulation insofern abgrenzen lassen, als bei der Emulation der Quelltext des Originalprogramms erhalten bleibt, die Hardware allerdings Ziel des Umschreibprozesses respektive der Übersetzung wird. Als Ergebnis von Umschreibprozessen stehen die nachfolgend betrachteten Ansätze dagegen zwischen den Polen des Verlustes ästhetischer und epistemischer Eigenheiten des Originals einerseits und der Entstehung neuer ästhetischer wie epistemischer Objekte andererseits. Das gilt auch für das Programm des Computermuseums der Stuttgarter Informatik, das zwar eine nahezu direkte Übersetzung von Lutz' Code in die 10 Jahre jüngere Assemblersprache des PDP-8 darstellt,[48] aber auf anderen Geräten als der ursprünglichen Z 22 läuft und dort beispielsweise im Hinblick auf die Geschwindigkeit ein anderes Verhalten aufweist.[49]

---

45 | Für ein Video der Reinszenierung vgl. auch: Computermuseum der Stuttgarter Informatik: »Stream vom 22.6.2022 - Thema: Theo Lutz - Stochastische Texte«.

46 | Höltgen: *Open History*, S. 136.

47 | Vgl. Loebel: *Lost in Translation*, S. 41; vgl. auch Höltgen: *Open History*, S. 247.

48 | Teilweise ist das bereits als simple Substitution der Befehlsbezeichner möglich, vgl. Computermuseum der Stuttgarter Informatik: »Stream vom 22.6.2022«, 1:05:30–1:07:40.

49 | Vgl. ebd., 57:45–59:00.

Mit Blick auf die durch die jeweilige Reimplementierung bedingte Materialität lässt sich feststellen, dass das Wechselspiel digitaler und analoger Komponenten zumeist zugunsten einer rein digitalen Komponente aufgelöst wurde: Lediglich bei der Reinszenierung des Codes im Computermuseum der Stuttgarter Informatik erfolgt die Eingabe über Lochstreifen und die Ausgabe über einen angeschlossenen Fernschreiber; in allen anderen Fällen werden die Texte auf einem Bildschirm angezeigt. Medium ist bei den letztgenannten zudem nicht ein zentraler Rechner, sondern der private Computer der Rezipient*innen: Auers und Montforts Ansätze sind webbasiert; der Code von eddeaddad läuft in JanusNode, einem kostenfreien Framework für die Erzeugung generativer Literatur. Das Computermuseum hat das Programm auf zwei historischen Rechnern implementiert – damit ist die Reinszenierung jedoch nicht unmittelbar für die Rezipient*innen zugänglich.

Auch der Ausgabeprozess der Texte eignet sich zur gesonderten Analyse: Bei Auer werden die Texte als Ganzes mit einer vorab definierten Menge an Zeilen ausgegeben, bei Montfort und eddeaddad werden die Sätze zeilenweise generiert. Dem ursprünglichen Generationsprozess am nächsten kommt die Stuttgarter Version, bei der jedes Zeichen einzeln gedruckt wird. Auch sind dabei verzögernde Effekte zu erkennen, die die Arbeit des Prozessors beim Aufruf des Zufallszahlengenerators anzeigen.[50] Die Maschine erscheint in den anderen drei Reimplementierungen daher als *Blackbox*, die nur bedingt Einblick in die ‚inneren Zustände' erlaubt. In keinem der Ansätze werden *Skeumorphismen* bemüht, – jene Elemente des Originalsystems, die in der Emulation lediglich mit dem Zweck der ästhetischen Nachbildung funktionaler Komponenten des Originalsystems auftreten[51] – wie sie häufig in Emulatoren Anwendung finden, weswegen eine große Distanz zur originalen materiellen Bedienoberfläche der Z 22 entsteht. Besonders bei Montfort ist das Interface sehr eingeschränkt: Neben der zeilenweisen Ausgabe des Textes auf schwarzem Hintergrund liegen nur weiterführende Links vor; der Ablauf des Programms kann nicht beeinflusst oder modifiziert werden. Im Unterschied dazu weist die Implementierung von Auer eine große Wahlfreiheit auf der Seite der Rezipient*innen auf, da hier bereits im Interface die Abänderung der Wörter des Substantiv-Lexikons vorgesehen ist.

Auf der Ebene des Zufallszahlengenerators gilt, dass Auer, eddeaddad und Montfort auf Lösungen zurückgreifen, die von den jeweils verwendeten Programmiersprachen beziehungsweise den jeweiligen Frameworks zur Verfügung gestellt werden. Bei Montfort findet sich dementsprechend zu Beginn des Python-Skripts die Zeile »from random import choice«, die den Zufallsgenerator ›choice‹ aus der in Python inkludierten Programmbibliothek ›random‹ im Skript initialisiert. Die Funktionsweise ist dabei, dass die Funktion ›choice‹ (pseudo-)zufällig ein Element aus einer Liste auswählt. Damit entfällt nicht nur der Umweg, über eine Zufallszahl auf die Indizierung der Speicherzellen zurückgreifen zu müssen, sondern auch die Verteilung der ›gezogenen‹ Elemente bei Lutz: Zwar liegt immer noch keine Gleichverteilung vor, die Verteilung ist bei Montforts Implementierung jedoch mit jeder Iteration eine andere; ohne die manuelle Angabe eines Seeds erscheint der Zufallsgenerator für die Rezipient*innen nicht nachvollziehbar deterministisch, da er sich im Fall der Python-Bibliothek auf die aktuelle Systemzeit als Seed stützt.

50 | Vgl. Computermuseum der Stuttgarter Informatik: »Stream vom 22.6.2022«, 57:45–59:00.

51 | Vgl. Loebel: *Lost in Translation*, S. 127.

In technischer Hinsicht ist anzumerken, dass auf den moderneren Systemen ein wesentlich größerer Speicher zur Verfügung steht, sodass Lutz' Beschränkungen auf kurze Wörter und auf eine begrenzte Zahl an Speicherzellen entfallen. Lutz' Lexikon wird trotzdem in allen Fällen direkt übernommen, wenngleich Anpassungen in der Schreibung der Umlaute und in der Kodierung des Spatiums am Ende der Wörter vorliegen.[52] So ist es nur im Fall von Auers Implementierung für die Rezipient*innen möglich, eigene Substantive und Adjektive zu definieren, die in der Generierung berücksichtigt werden.[53] Schlussfolgernd lässt sich für die Implementierungen festhalten, dass sie auf der Ebene der Programmierung und der Technik stark von Lutz' Code abweichen, nicht zuletzt, da sie einen »Zeichenwechsel«[54] im Sprung zwischen mehreren Programmiersprachen und Maschinen vollziehen müssen. Grundsätzlich ist zu erkennen, dass die Reimplementierungen sich nach dem publizierten Text richten und das ursprüngliche Programm nicht reflektieren. Auf der Ebene des Algorithmus jedoch bleiben sie dennoch nah an der – wie auch in Lutz' Artikel skizzierten – Idee von Lutz: Ein Zufallsgenerator wählt aus einer gegebenen Menge an Substantiven, Adjektiven, Artikel und Konjunktionen und generiert so Zeilen bestehend aus zwei kurzen Sätzen. In der Übersetzung in eine andere Sprache und als Resultat eines Umschreibprozesses stellen sie eine aktualisierende Rezeption dar, die jedoch eigene digitale Objekte erzeugt – auf epistemischer wie auf ästhetischer Ebene. Dabei kodieren auch sie in ihren Programmen ihre Produktionsbedingungen: Die verfügbare Technik wird referenziert und wie im Falle des Zufallsgenerators direkt aufgerufen. Den Erhalt der Charakteristika von Lutz' Implementierung können sie daher jedoch nicht leisten. Im letzten Kapitel dieser Analyse findet aus diesem Grund eine Reflexion über das Potenzial der Emulation des originalen Codes auf einem Emulator der Z 22 statt.

52 | Montfort übersetzt das Lexikon zudem ins Englische.

53 | Der originale Output ist hier bei Klick auf die Schaltfläche »Lutzen original (ohne Fehlerkorrektur)« abzurufen. Beim Abgleich dieser ›originalen‹ Ausgabe fällt jedoch auf: Auer reflektiert zwar die Fehler in der Ausgabe der Z 22 in Form von Rechtschreib- und Satzzeichenfehlern, erfasst aber nicht die Charakteristika und Fehler des Programms, beispielsweise die abweichenden Formen »HAS« und »JEEER«, obwohl ihm die originale Ausgabe vorliegt: Er verlinkt ein Faksimile des Fernschreiberausdrucks auf der Website (vgl. Johannes Auer: »Theo Lutz (23.7.1932 - 31.1.2010): Stochastische Texte (1959)«, 2005, https://auer.netzliteratur.net/0_lutz/lutz_original.html (zuletzt eingesehen am 06. August 2023)).

54 | Höltgen: *Open History*, S. 247.

4 »Schaltbild zeigen«: Zum Einsatz von Emulation in der Erhaltung der *Stochastischen Texte*

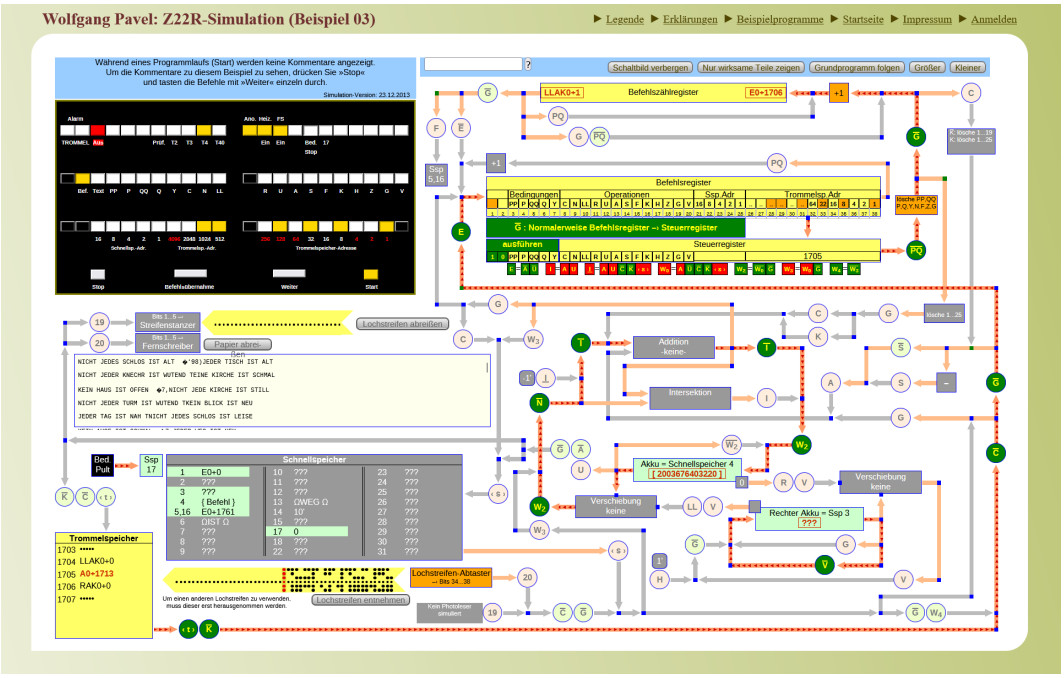


Abb. 4: Screenshot des Emulators von Pavel mit einer zeichengenauen Abschrift von Lutz' Programm für die Stochastischen Texte.

Für die Zuse Z 22 gibt es zwei Emulatoren im Internet: von Wolfgang Pavel[55] und von Norbert Kehrer.[56] Beide sind in der Lage, Programme, die in der Maschinensprache der Z 22 oder im Freiburger Code geschrieben sind, nach dem *CCITT-2* in (virtuelle) Lochstreifen zu konvertieren, die Prozesse innerhalb der Maschine zu simulieren und die Fernschreiberaufrufe auf den Bildschirm auszugeben. Damit handelt es sich im Gegensatz zu den oben besprochenen Ansätzen um dezidierte Emulationen, die es erlauben, digitale Objekte, die auf der Z 22 entwickelt wurden, in ihrer ursprünglichen digitalen Form zu belassen,[57] ohne den Code verändern zu müssen.

Im Rahmen des vorliegenden Beitrags konnte anhand der im DLA Marbach erhaltenen Programmausdrucke eine zeichengenaue Abschrift des Programms für die *Stochastischen Texte* angefertigt werden,[58] die anschließend als Eingabe für den Emulator von Pavel genutzt wurde (vgl. Abb. 4). Auf diesem Programmdurchlauf aufbauend sollen nun abschließend einige Einblicke in das Potenzial und die Grenzen von Emulation für die Erschließung digitaler Literatur erfolgen.

Auffallend ist zunächst das Interface des Emulators, das dem Bedienpult der Z 22 nachgebildet ist: Mit der Verwendung solcher *Skeumorphismen* ist es notwendig, über eine rudimentäre Kenntnis der Bedienung einer Z 22 zu verfügen, bevor Programme in den Speicher geladen und gestartet werden können. Die sehr reduzierte Sicht auf die

---

55 | Wolfgang Pavel: »Z22R-Simulation«, 23.Dezember 2013, https://www.wpavel.de/zuse/simu/simu.php (zuletzt eingesehen am 06. August 2023).

56 | Norbert Kehrer: »Zuse Z22 Emulator«, Oktober 2018, https://norbertkehrer.github.io/z22.html (zuletzt eingesehen am 06. August 2023).

57 | Vgl. Thibodeau: »Overview of Technological Approaches to Digital Preservation«, S. 5.

58 | Die Bereitstellung dieser Transkription ist seitens des Autors dieses Beitrags in Vorbereitung.

inneren Zustände der Maschine wird im Emulator erweitert: Es ist möglich, das vollständige Schaltbild der Maschine anzuzeigen, das Einblicke in die Belegung der einzelnen Speicherzellen sowie in den Fluss einzelner Ströme in der Verkabelung gibt. Wie oben angegeben, ist dieser Einblick in der historischen Maschine nicht möglich; diese Funktionalität lässt sich im Sinne Höltgens als »Fehler« der Emulation betrachten – insbesondere als »error of addition«,[59] der den Funktionsumfang im Verhältnis zum Originalsystem vergrößert.[60]

Die Ausgabe findet auf dem Bildschirm des Computers statt; die historische Materialität des Fernschreiberausdrucks entfällt. Auch die Rezeptionssituation ist eine andere: War die Stuttgarter Z 22 räumlich fest installiert und die Nutzung an die institutionelle Öffentlichkeit gebunden,[61] so ist die Situation der Nutzer*innen vor dem PC flexibel und privat. Mit Thibodeau lässt sich daher argumentieren, dass das Objekt auf physischer und auf logischer Ebene erhalten bleibt, als konzeptuelles Objekt jedoch ein anderes ist. Das spiegelt sich in der Nutzung von Emulatoren wider: »Emulators tend to be a tool for accessibility, rather than a tool to preserve the intact object and its original context.«[62] Die angesprochene ›accessibility‹ als Hauptzweck von Emulatoren zu sehen, bietet großes Potenzial für deren wissenschaftliche Nutzung. Kaltman, Osborn und Wardrip-Fruin demonstrieren anhand des Videospiels *Doom*, wie Emulatoren in einen wissenschaftlichen Aufsatz eingebunden werden können, um spezifische Spielelemente interaktiv abrufbar zu machen.[63] Indem der Emulator einen konkreten Zustand der Maschine simulieren kann, erlaubt er, Forschungsergebnisse zu reproduzieren und sie am operativen Objekt zu *demonstrieren*.[64] Eben diese neuralgische Funktion erfüllt der Durchlauf von Lutz' Programm im Emulator: Das Verhalten des Programms in seinen (Zwischen-)Ausgaben kann nicht nur gesamt, sondern befehlsweise repliziert und so Ausgangspunkt weiterer Analysen werden.

Das Programm startet und generiert bei Verwendung des gleichen *Seed*-Wertes 12345678 Zeichen für Zeichen die historische Ausgabe der Z 22 bei Lutz. Dabei kann in der erweiterten Schaltbild-Ansicht die Belegung der Speicherzellen mit Wörtern, Zahlen und Befehlen überprüft werden. Auch die werkimmanenten algorithmischen Schwachstellen des Programms bleiben auf diese Weise erhalten: In der Emulation sind die fehlerhaften Formen »FREMDE« und »SCHLOS« zu finden. Eine neue Fehlerquelle ergibt sich jedoch trotz der zeilengenauen Abschrift des Programms bei der Setzung der Konnektoren zwischen den beiden kurzen Sätzen. Hier gibt der Emulator nicht lesbare Sonderzeichen aus. Ob an dieser Stelle das Programm von Lutz, der Emulator[65] oder die

---

59 | Höltgen: *Open History*, S. 244.

60 | Vgl. auch Loebel: *Lost in Translation*, S. 58–60.

61 | Vgl. Hoffmann: »Eine Maschine und ihr Betrieb«, S. 122.

62 | Dor: »Emulation«, S. 26.

63 | Vgl. Eric Kaltman, Joseph Osborn u. Noah Wardrip-Fruin: »From the Presupposition of Doom to the Manifestation of Code. Using Emulated Citation in the Study of Games and Cultural Software«. In: *Digital Humanities Quarterly* 15.1 (2021), http://www.digitalhumanities.org/dhq/vol/15/1/000501/000501.html (zuletzt eingesehen am 06. August 2023).

64 | Zur Relevanz der Demonstration für die Computerarchäologie vgl. Höltgen: *Open History*, S. 80.

65 | Für die meisten Emulatoren ist zusätzlich anzumerken, dass es sich um Privatprojekte handelt, weswegen anzunehmen ist, dass »wohldefinierte Standards und Arbeitsmethoden« (Loebel: *Lost in Translation*, S. 146) entfallen. Auch für Pavels Emulator finden sich über die Angaben zur historischen Programmiersprache hinaus wenige Informationen zur Entwicklung und Versionierung des Emulators auf der Website.

Kompatibilität von beidem verantwortlich für das Verhalten der virtuellen Maschine ist, lässt sich im Rahmen dieser Arbeit nicht mit Sicherheit feststellen: »You will never know if an obscure game you load within an emulator is correctly rendered, if you don't have the real functionality of the original hardware.«[66] Dieses Problem ist für die wissenschaftliche Erschließung historischer Programme von besonderer Relevanz, stehen doch für ältere Hardware zumeist keine operationalen Geräte zur Verfügung, anhand derer die Funktionalität abgeglichen und gewährleistet werden kann. Claus Pias fasst die paradoxen Schwierigkeiten dieser Situation zusammen:

> Um nämlich Überlieferungsfehler einer Emulation überhaupt festzustellen, sind intensive Vergleichstests des emulierten Systems mit dem Originalsystem notwendig, die aber nur durchgeführt werden können, solange es das Originalsystem noch in lauffähigem Zustand gibt – also in der Kopräsenz dessen, was es für eine Zukunft noch lesbar zu machen gilt.[67]

Für die Z 22 ist zum gegenwärtigen Zeitpunkt kein vollständig funktionsfähiges Exemplar erhalten – der Emulator Pavels lässt sich nur auf Basis historischer Zeugnisse einerseits und mathematisch-abstrahierter Modellierung andererseits überprüfen. Für die wissenschaftliche Nutzung dieses Emulators bedarf es daher eines »›starken‹ Editorensubjekts«,[68] das in der Lage ist, diesen Abgleich mit dem Originalgerät auf historischer und computerarchäologischer Ebene zu leisten und die Ergebnisse einzuordnen. Ferner rückt damit auch die Emulation selbst in die Nähe eines interpretatorischen Aktes,[69] die als Modell für das Originalverhalten eine heuristische Annäherung zur Verfügung stellt.

Das Verhalten des Programms im Emulator ermöglicht verschiedene Einblicke: *Einerseits* wird – wie bereits in Abschnitt 2 gezeigt – deutlich, dass die publizierte Fassung der *Stochastischen Texte* tatsächlich und replizierbar von der Fernschreiberausgabe der Z 22 divergiert. *Andererseits* zeigt sich das Potenzial zu einem erweiterten Werkverständnis:[70] Durch die Interaktion mit dem historischen Code – vom Abtippen der Befehle bis zum Einlegen des virtuellen Lochstreifens – erhält dieser den Status eines gemachten Werks, das gelesen, verarbeitet und dokumentiert werden kann. Im Emulator erlangt der Code auf diese Weise Unabhängigkeit von der späteren Publikation der *Stochastischen Texte* durch Lutz und von den späteren Reimplementierungen; Er kann so (computer-)archäologisch geborgen und als Artefakt rekonstruiert werden.[71]

Es lässt sich resümieren, dass die bisherigen Reimplementierungen die *Stochastischen Texte* lediglich auf der Ebene des Algorithmus beziehungsweise als Konzept aufgreifen. Der Umschreibprozess in eine neue Programmiersprache generiert neue digitale Objekte (mit jeweils eigenen physischen, logischen und konzeptuellen Eigenschaften), die nicht in der Lage sind, die *Stochastischen Texte* als physisches, logisches oder konzeptuelles Objekt zu archivieren. Stattdessen entstehen in diachroner Perspektive neue Formen der Autorschaft unter dem Vorzeichen sich stetig verändernden Produktionsbedingungen generativer Literatur. Weitere Forschung zu diesen Formen der Autorschaft ist nötig: Diese in ihrer vollständigen Komplexität zu analysieren bedarf eines erweiterten Werkzeugkastens, der *einerseits* vermag, die Programme als textuelle Zeugnisse zu

---

66 | Dor: »Emulation«, S. 27.

67 | Pias: »Medienphilologie und ihre Grenzen«, S. 377.

68 | Ebd.

69 | Ebd., S. 384.

70 | Zum Werkcharakter von Lutz' Code vgl. auch Bernhart: »Beiwerk als Werk«, S. 204.

71 | Vgl. Höltgen: *Open History*, S. 71.

erschließen und zu vergleichen, sowie *andererseits* die Produktionsbedingungen in den Analyseprozess einzuschließen. Damit muss die Arbeit im *Dazwischen* informatischer und literarischer Forschung stehen. Im vorliegenden Beitrag konnte die Eignung computerarchäologischer Arbeitsweisen anhand Lutz' frühen Werkes aufgezeigt werden: Mit einem dezidierten Emulator ist es möglich, die physische und logische Ebene des Objekts in ihrer operationalen Form zu erhalten. Um jedoch die Divergenzen und Fehler auf der konzeptuellen Ebene zu erfassen, zu dokumentieren und für die Anschlussforschung nutzbar zu machen, muss schließlich auch der Werkzeuggebrauch des Emulators zur interpretativen Rezeptionsleistung werden: Der Code des Autors ist die Geburt der Computerarchäologie.

Literaturverzeichnis

Auer, Johannes: »Theo Lutz (23.7.1932 - 31.1.2010): Stochastische Texte (1959)«, 2005, https://auer.netzliteratur.net/0_lutz/lutz_original.html (zuletzt eingesehen am 06. August 2023).

Bernhart, Toni, Christian Corti u. Klemens Krause [= Computermuseum der Stuttgarter Informatik]: »Theo Lutz. Stochastische Texte. Vom Freiburger Code zum Stuttgarter Code«, 2022, https://www.fo5.uni-stuttgart.de/informatik/fachbereich/computermuseum/projekte-und-referenzen/theo-lutz-stochastische-texte/ (zuletzt eingesehen am 06. August 2023).

Bernhart, Toni: »Beiwerk als Werk. *Stochastische Texte* von Theo Lutz«. In: *editio* 34.1 (2020), S. 180–206, DOI: 10.1515/editio-2020-0010.

Bernhart, Toni: »Theo Lutz auf Zuse Z 22: *Stochastische Texte* (1959). Präliminarien einer Edition«. In: Laura Auteri u. a. (Hg.): *Wege der transkulturellen Germanistik. Akten des XIV. Kongresses der Internationalen Vereinigung für Germanistik (IVG).* Bd. 8. Bern 2022, S. 139–151 (Jahrbuch für Internationale Germanistik, Beihefte).

Computermuseum Der Stuttgarter Informatik: »Stream vom 22.6.2022 - Thema: Theo Lutz - Stochastische Texte«, *YouTube*, 11. Juli 2022, https://www.youtube.com/watch?v=9EfzNPxs82M (zuletzt eingesehen am 06. August 2023).

Dor, Simon: »Emulation«. In: Mark J. P. Wolf u. Bernard Perron (Hg.): *The Routledge Companion to Video Game Studies.* New York 2013, S. 25–31.

Eddeaddad: »2 Lutz Fragments«, 22. Dezember 2010, https://gnoetrydaily.wordpress.com/2010/12/22/2-lutz-fragments/ (zuletzt eingesehen am 06. August 2023).

Foucault, Michel: *Archäologie des Wissens.* Übers. v. Ulrich Köppen. Frankfurt a. M. 1981.

Höltgen, Stefan: *Open History. Archäologie des Retrocomputings.* Berlin 2022.

Kittler, Friedrich: »Es gibt keine Software«. In: Ders.: *Draculas Vermächtnis. Technische Schriften.* Leipzig 1993, S. 225–242.

Hoffmann, Christoph: »Eine Maschine und ihr Betrieb. Zur Gründung des Recheninstituts der Technischen Hochschule Stuttgart (1956–1964)«. In: Ders., Barbara Büscher u. Hans-Christian von Herrmann (Hg.): *Ästhetik als Programm. Max Bense/Daten und Streuungen.* Berlin 2004, S. 119–129.

Kaltman, Eric, Joseph Osborn u. Noah Wardrip-Fruin: »From the Presupposition of Doom to the Manifestation of Code. Using Emulated Citation in the Study of Games and Cultural Software«. In: *Digital Humanities Quarterly* 15.1 (2021), http://www.digitalhumanities.org/dhq/vol/15/1/000501/000501.html (zuletzt eingesehen am 06. August 2023).

Kehrer, Norbert: »Zuse Z22 Emulator«, Oktober 2018, https://norbertkehrer.github.io/z22.html (zuletzt eingesehen am 06. August 2023).

L'Ecuyer, Pierre: »History of Uniform Random Number Generation«. In: W. K. V. Chan u. a. (Hg.): *Proceedings of the 2017 Winter Simulation Conference*, Las Vegas 2017, S. 202–230, DOI: 10.1109/WSC.2017.8247790.

Link, David: »LoveLetters_1.0«. 2009. http://www.alpha60.de/art/love_letters/ (zuletzt eingesehen am 06. August 2023).

Link, David: *Archaeology of Algorithmic Artefacts.* Minneapolis 2016.

Loebel, Jens-Martin: *Lost in Translation. Leistungsfähigkeit, Einsatz und Grenzen von Emulatoren bei der Langzeitbewahrung digitaler multimedialer Objekte am Beispiel von Computerspielen.* Berlin 2013.

Lutz, Theo: »Kybernetik, Struktur und Simulation«. In: *Soziale Welt* 16.1 (1965), S. 27–45.

Lutz, Theo: »Stochastische Texte« [1959]. In: Barbara Büscher, Christoph Hoffmann u. Hans-Christian von Herrmann (Hg.): *Ästhetik als Programm. Max Bense/Daten und Streuungen.* Berlin 2004, S. 164–169.

Lutz, Theo: [Fernschreiberausdruck mit der handschriftlichen Angabe »Programm für STOCHASTISCHE TEXTE. 2. Auflage. 29. Juli 1959«]. *Deutsches Literaturarchiv Marbach* (DLA, A: Lutz, Theo, Kasten 2, Mappe 3) (zitiert als STP).

Lutz, Theo: [Fernschreiberausdruck ohne Titel; Incipit: »NICHT JEDER BLICK IST NAH«]. *Deutsches Literaturarchiv Marbach* (DLA, A: Lutz, Theo, Kasten 2, Mappe 3) (zitiert als STT).

Montfort, Nick: »Stochastic Texts«, 2014, https://nickm.com/memslam/stochastic_texts.html (zuletzt eingesehen am 06. August 2023).

Pavel, Wolfgang: »Zuse Z22. Dokumentation und Simulation«, 23. Dezember 2013, https://www.wpavel.de/zuse/doku/doku_simu.php (zuletzt eingesehen am 06. August 2023).

Pavel, Wolfgang: »Z22R-Simulation«, 23. Dezember 2013, https://www.wpavel.de/zuse/simu/simu.php (zuletzt eingesehen am 06. August 2023).

Pias, Claus: »Medienphilologie und ihre Grenzen«. In: Friedrich Balke u. Rupert Gaderer (Hg.): *Medienphilologie. Konturen eines Paradigmas*. Göttingen 2017, S. 365–385.

Reiter, Nils: »Sitzung 1. Einführung, Ablauf des Kurses, Organisatorisches« [Lehrmaterialien für die Übung *Reenactment historischer Rechner*], 2022, https://lehre.idh.uni-koeln.de/site/assets/files/3972/session-01.pdf (zuletzt eingesehen am 06. August 2023).

Reiter, Nils: »Sitzung 7. Zufall« [Lehrmaterialien für die Übung *Reenactment historischer Rechner*], 2022, https://lehre.idh.uni-koeln.de/site/assets/files/3972/session-07.pdf (zuletzt eingesehen am 06. August 2023).

Thibodeau, Kenneth: »Overview of Technological Approaches to Digital Preservation«. In: Council on Library and Information Resources (Hg.): *The State of Digital Preservation: An International Perspective*. Washington, D. C. 2002, S. 4–31.

Zuse KG: *Programmgesteuerte Elektronische Rechenanlage Zuse Z 22 und Z 22 R. Programmierungsanleitung*. Bad Hersfeld 1960.

DOI: 10.17879/86988447019