

# Assistenzsysteme

Mimetische Praktiken verteilter  
Autorschaft zwischen Mensch und Maschine

Markus Krajewski, Universität Basel,

local version 0.75:2019-02-15,  
[github.com/nachsommer/VersionsKontrolle](https://github.com/nachsommer/VersionsKontrolle): e93461-dirty

erscheint in: Friedrich Balke und Elisa Linseisen (Hrsg.),  
*Mimesis Expanded*, Wilhelm Fink Verlag, Paderborn, 2019

## Inhaltsverzeichnis

<b>0 Versionierungen – A Brief Introduction</b>	<b>5</b>
<b>1 Verzweigen</b>	<b>9</b>
<b>2 Kopieren</b>	<b>11</b>
<b>3 Abweichen</b>	<b>13</b>
<b>4 Verschmelzen</b>	<b>15</b>
<b>5 Coda: Assistenzsysteme, Quellcodekritik und Auto(r)Korrektur</b>	<b>15</b>

- Ich fange mal mit der Gegenwart und ein paar losen Enden an, die es zu verknüpfen und in eine Textur zu bringen gilt.... [●]

Ihr seht hier zahlreiche lose Enden, die nicht nur die Gegenwart des Schreibens repräsentieren, sondern auch die vielen möglichen Zukünfte von kollektiver Autorschaft darstellen – 2018. Denn was dieses Bild zeigt, ist ein evolutionäres Schema oder ein Wissensbaum, der die verschiedenen Derivate, also Abspaltungen von frei entwickelten Betriebssystemen aus den ursprünglichen GNU/Linux-Distributionen heraus dokumentiert. Jede dieser Linien repräsentiert Tausende, oftmals Millionen Zeilen Code, die allesamt in einem fein abgestimmten, selbst über beinahe 50 Jahre entwickelten Milieu von kollektiver Autorschaft entstanden sind.

Nicht dass Ihr denkt, dass das schon alles ist: GNU/Linux selbst ist ja nur ein Derivat von Unix, aus dem auch Android, macOS und eigentlich alles andere Wichtige an Betriebssystemen ausser Windows hervorgegangen ist. ●

- Worum geht's? – Einerseits geht's mir darum, einen längeren historischen Bogen zu schlagen, um die medialen Praktiken der Softwareentwicklung zu historisieren und so einer Art Mediengeschichte der Software-Studies Vorschub zu leisten und auf diese Weise der Geschichtsvergessenheit der *software studies* ein wenig entgegenzuarbeiten, um die gegenwärtige Praktik kollektiver Autorschaft zu ihren historischen Wurzeln und Entwicklungslinien zurück zu verfolgen.
- Andererseits geht's mir darum, die im Folgenden vorzustellenden medialen Praktiken des Verzweigens, Kopierens, Verschmelzens als generische Verfahren zu beschreiben, denen – und das dürfte in unserem Zusammenhang hier wenig überraschen – eine spezifisch produktive oder kreative Komponente eignet.
- Dabei möchte ich *en passant* zwei Konzepte entwickeln, zum einen den eher methodischen Begriff der Quellcodekritik, also einen Weg, um Software-Studien um eine historiographische Komponente zu erweitern, zum anderen den Begriff der Assistenzsysteme, der als ein medienmimetisches Konzept zentral für die hier gewählten Beispiele und Szenarien einzustufen ist und sich als der Zusammenhang erwiesen hat, in dem sich Autor-Lektor-Konstellationen ebenso wiederfinden wie subalterne Sekretärsschriftsätze, aber eben auch kollektive Schreibprozesse, die im Zusammenspiel von Menschen (= Software-Entwickler) und speziellen Schreibumgebungen vollzogen werden, also in und mit IDEs (Integrated Development Environments) und Versionskontrollsystemen wie github, dazu gleich mehr.

Was versteht man in gängiger Diktion unter Assistenzsystemen? Der Begriff findet derzeit vor allem im Kontext des Individualverkehrs seine Verwendung. Assistenzsysteme überwachen, etwa beim Steuern eines Automobils, dass der Fahrer nicht unverhofft aus der Spur gerät. Oder sie dienen dazu, bei knifflig-kurzen Parklücken, dass ein Einparken ohne Lackschäden auf Knopfdruck und automatisch erfolgt. Assistenzsysteme können – neben der Sorge, nicht aus der Spur zu geraten – sowohl Patienten »bei Alltagshandlungen unterstützen als auch Wissenschaftler bei der Auswahl und dem Einsatz von Analysemethoden, Sportler im Training oder Arbeiter bei der Montage. Methodisch setzen Assistenzsysteme vielfältige Techniken aus den Bereichen der Mensch-Maschine-Interaktion, der sensorbasierten Zustandsschätzung und der künstlichen Intelligenz ein.«<sup>1</sup> Ich möchte diesen Begriff aufnehmen, um ihn aus der höchst zweifelhaften Beschränkung auf ungelenke Lenksysteme oder die exklusive Verwendung in der Informatik herauszulösen. Vielmehr geht es darum, auch dieser Bezeichnung so etwas wie historische Tiefenschärfe zu verleihen, um nachzuweisen, dass es nicht nur im Verkehrsgeschehen, sondern vor allem in ungleich weiter zurückliegenden, historischen Zusammenhängen, sei es beim Handel, sei es im Haushalt, oder sei es im Kontext literarischer Produktion immer schon auf die *systematischen, das heisst regelgeleiteten Kooperationen zwischen Assistenz und Akteur* angekommen ist.

---

1. [www.informatik.uni-rostock.de/forschung/schwerpunkte/intelligente-assistenz/](http://www.informatik.uni-rostock.de/forschung/schwerpunkte/intelligente-assistenz/)

Die Geschichte verteilter Autorschaft lässt sich unschwer als die lange Genealogie des Schreibens überhaupt identifizieren, steht doch an den Anfängen der Schrift kein Individuum, kein Autor im Sinne seiner goethezeitlichen Ausdifferenzierung zur männlichen Verkörperung einer Genie-Ästhetik, sondern immer schon ein Kollektiv, sei es nun, um den Pentateuch zu verfassen oder das Gilgamesch-Epos, oder sei es, um unter einem mutmasslichen Kollektivsingular wie Homer Verse auf Papyrus zu fixieren. Texte sind *qua definitionem* Verbindungen loser Enden, die von mehr als einer Person bearbeitet werden. Dies gilt umso mehr, wenn man einen die Philologie übersteigenden Textbegriff zugrunde legt, der ebenso operative Schriften, also Code und seine Entwicklung in Softwareprojekten betrachtet, die inzwischen – schon infolge ihrer Komplexität – zumeist von mehr als einer Person vorangetrieben werden.

Wenn nun aber mehrere, viele, ja tausende Entwickler an ein und demselben Code arbeiten, erfordert es besondere Maßnahmen, um die Konsistenz dieser kollaborativen Autorschaft sicherzustellen. Was ich im Folgenden skizzieren möchte [●], ist daher zunächst ein Blick hinter die Kulissen, wie grossangelegte kollektive Schreibprojekte mit Hilfe von sog. Versionsverwaltungen und ihren grundlegenden Befehlen wie *branch*, *diff* und *merge* medientechnisch organisiert sind und welchen, wenn man so will, Befehlssätzen und Befehlsketten sie gehorchen, bevor ich etwas allgemeiner das Konzept einer Quellcodekritik vorstellen möchte. Die Beschreibung der gegenwärtigen Quellcodeentwicklung werde ich allerdings anhand eines historischen Szenarios zu verdeutlichen versuchen, um damit gleich eine entsprechende genealogische Linie zu erhalten, die jene Praktiken des Codings auf ihre historischen Vorläufer und Grundlagen zurückführt.

Meine genealogische Rekonstruktion stützt sich zunächst auf einige exemplarische Geschichten oder historische Szenarien, die sich mehr oder minder synchron ereignen, und zwar um 1780, zum einen im Frankreich des Ancien Regime, zum zweiten in der Hauptstadt des Heiligen Römischen Reichs Deutscher Nation, in Wien, und aber natürlich auch hier, in Weimar mit seiner goethezeitlichen Perfektionierung von Autorschaft. Daneben liessen sich aber auch ganz andere Szenarien in London 1916 und anderenorts anführen.

Das erste Szenario zielt nun zunächst darauf zu verdeutlichen, was da eigentlich ganz praktisch geschieht, wenn mehrere Autoren gemeinsam an Texten resp. Code schreiben. Dazu versetzen wir uns in das vorrevolutionäre Frankreich und *gleichzeitig* wieder unmittelbar in die Gegenwart des kollektiven Schreibens. Ich versuche, in aller gebotenen Kürze, drei Techniken kollektiver Autorschaft zu erläutern, die wesentlich für die kollaborative Code-Entwicklung sind. Es handelt sich um *branch*, *diff* und *merge*.

## 0 Versionierungen – A Brief Introduction

Die eigentümlichen Imperative *branch*, *diff*, *merge* sind nicht nur unschuldige (Stamm-)Formen englischer Verben, sondern finden ebenso in einem informatischen Kontext, konkret bei der sogenannten Versionsverwaltung oder zu Englisch der *version control*, Verwendung. Derartige Versionsverwaltungen kommen einerseits im Hintergrund von organisatorischen Maßnahmen auf Betriebssystemen zum Einsatz, also etwa bei der eingebauten Backup-Funktion auf macOS namens ›TimeMachine‹. Andererseits bilden sie das Kernstück sowohl für lokale als auch für zentrale oder gar global verteilte Softwareentwicklungsprojekte, wo Entwickler an verteilten Orten gleichzeitig an demselben Code arbeiten, dessen Änderungen demzufolge zeichengenau und auf die Mikrosekunde exakt protokolliert werden und nachvollziehbar bleiben müssen. Das bekannteste Forum dieser Art dürfte derzeit die von Linus Torvalds initiierte Plattform *github* sein.

Das Ziel der Kodeentwicklung ist dabei – leicht idealisiert – wie bei einer konventionellen Textproduktion zu verstehen, wo ja am Ende eine Abfolge von Zeichen entsteht, die – bei hinreichendem Interesse möglicher Leser und ausreichender Schreibkunst der Autoren – vom ersten bis zum letzten Zeichen rezipiert wird. Ähnlich akribisch kann man sich im informatischen Kontext als zentrale Entscheidungsgewalt den sog. *compiler* vorstellen, also jene Instanz bei der Programmierung, die den vorbereiteten Code in einer beliebigen (höheren) Programmiersprache in den allein ausführbaren Binärcode der Maschinensprache übersetzt. Auch bei diesem Übersetzungsprozess wird jedes Zeichen, jeder Befehl, jede Schleife, jede Datenstruktur sequentiell, von vorne bis hinten, linear eingelesen, validiert und interpretiert. Man muss sich den *compiler* als einen Meister des *close reading* vorstellen.<sup>2</sup> Um also einen Programmcode ›lauffähig‹ zu machen, muss er einmal linearisiert, das heisst jeder der zahlreichen Befehle muss Zeile für Zeile ausgewertet werden. Der *compiler* zieht die Teile des Programmcodes aus unterschiedlichsten Bereichen zusammen, aus entlegenen Programmbibliotheken ebenso wie aus offenen Quellen, die dezentral im Internet vorgehalten werden, um alles in eine lineare Abfolge zu bringen. Nun kann es aber vorkommen, dass über die genaue Abfolge der Befehle, Programm- und Datenstrukturen Uneinigkeit herrscht innerhalb der Gemeinschaft der Codeentwickler eines bestimmten Projekts. Angenommen, ab Codezeile 13'531 stehen folgende Befehle [●]:

Weiter angenommen, dass bezüglich der Codezeile 13'541 eine Diskussion in der weltweiten Entwicklergemeinschaft entbrennt, weil der Entwickler **Egmont** der Meinung ist, hier müsse statt ›Schloß Cormatin‹ vielmehr ›Schloß Chambord‹ stehen, da infolge eines Unwetters im Südosten von Paris die Straßen am 6. August 1781 nach Burgund unwegsam waren, so dass kein Postillion seine Briefe zustellen konnte, was wiederum

---

2. **krajewski+vismann:2009.**

```

13531 leseBrief("Cécile Volanges", "Sophie Carnay",
13532         gegeben("Paris",1781-08-03));           // 1. Brief
13533
13534 leseBrief("Marquise de Merteuil", "Vicomte de Valmont",
13535         gegeben("Paris",1781-08-04));           // 2. Brief
13536
13537 leseBrief("Cécile Volanges", "Sophie Carnay",
13538         gegeben("Paris",1781-08-04));           // 3. Brief
13539
13540 leseBrief("Vicomte de Valmont", "Marquise de Merteuil",
13541         gegeben("Schloß Cormatin",1781-08-05)); // 4. Brief
13542
13543 leseBrief("Marquise de Merteuil", "Vicomte de Valmont",
13544         gegeben("Paris",1781-08-07));           // 5. Brief

```

dazu führte, dass infolge der üblichen Brieflaufzeiten die Antwort der Marquise de Merteuil niemals am 7. August hätte geschrieben werden können. Das Programm sei also, so **Egmont**, an dieser Stelle fehlerhaft. Der Entwickler **Richard**, auf den der ursprüngliche Code zurückgeht, kann sich mit diesem Argument allerdings nicht einverstanden erklären und beharrt auf seiner anfänglichen Lokalisierung des Briefs des Vicomte de Valmont auf Schloß Cormatin. Der Konflikt zwischen den beiden Kontrahenten bleibt ungelöst und der Programmcode wird an dieser Stelle kurzerhand verzweigt [●Abb. 1], das heisst mit Hilfe des Befehls *branch* gelingt es, den Code zu duplizieren, um beide Varianten parallel zueinander existieren zu lassen. **Egmont** schert also an dieser Stelle aus dem linearen Ablauf der Befehlskette aus, indem er einfach seine eigene Variante unter dem Etikett eines neuen *branch* (Zweigs) der Allgemeinheit zur Verfügung stellt.

Beide Stränge sind nach wie vor für alle Entwickler sichtbar und nahezu identisch, bis auf die kleine Abweichung von **Richard**, der seinen *source code* nach den eigenen Vorstellungen abgeändert hat. Um hier den Überblick nicht zu verlieren, kann man mit dem kleinen Hilfsprogramm *diff* den Befehl erteilen, sich die Unterschiede in den beiden Quellcodes anzeigen zu lassen. Das sieht dann so aus [●Abb. 2]:

*diff* macht also den kleinen Unterschied sichtbar. Das Programm hebt die Abweichungen zweier in weiten Teilen identischen Dokumente hervor, oder um es – leicht abweichend – mit einem informatischen Begriff zu sagen: *diff* macht die Deltas innerhalb des Codes sichtbar, oder um es – erneut leicht abweichend – mit einem philosophischen Begriff zu sagen: *diff* führt die *différence* zwischen Original und Kopie vor. – Das Programm *diff* wurde in den frühen 70er Jahren von Douglas McIlroy an den Bell Labs in New Jersey geschrieben. Die Denkfigur *différence* wurde in den frühen 70er Jahren von Jacques Derrida an der ENS in Paris entwickelt.

Nun könnten sich beide Zweige von ein und demselben Programm parallel zueinander weiterentwickeln, sich dabei zunehmend unterscheiden, in mehr als nur einer Zeile voneinander abweichen, so lange bis es zwei

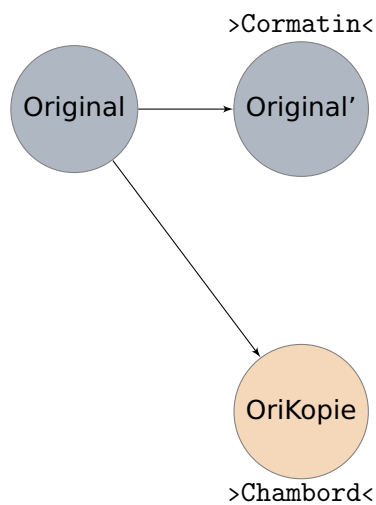


Abbildung 1: Varianten einer Entwicklung

ganz unterschiedliche Programme geworden sein werden, also so, wie bei den Linux-Derivaten aus der Anfangsgraphik mit zahlreichen losen Enden. Doch diese auseinanderstrebende Bewegung der beiden Teile wird in unserem Beispiel durch einen neuen Forschungsstand jäh unterbunden. In einem Konvolut im Nachlass von Pierre-Ambroise-François Choderlos de Laclos taucht der Hinweis auf, dass es sich bei dem von ihm in der Druckfassung bewusst als leere Variablen belassenen, mit Auslassungspunkten gekennzeichneten Ortsangaben von Valmonts Aufenthaltsort um das Schloß Bussy-Rabutin gehandelt habe, wie der Nutzer **Oliva** glaubhaft machen kann. Der Konflikt ist damit durch eine neue archivalische Evidenz geschlichtet, die beiden falschen Angaben ›Chambord‹ und ›Cormatin‹ gilt es zu ersetzen durch ›Schloß Bussy-Rabutin‹, um damit die beiden separaten Stränge wieder zusammenzuführen [Abb. 3 ●]. Diese Konvergenzbewegung wird durch den Befehl *merge* erreicht, der die beiden konkurrierenden Darstellungen wieder vereint, indem zunächst einer der beiden Versionen in Programmzeile 13'541 der Vorzug gegeben wird, um den Code dann mit der neuen Erkenntnis und ergänzt um einen Kommentar erneut zu einem einzigen Zweig zu fusionieren. Dieses Verfahren, das auch als *three-way-merge* bezeichnet wird, erfreut sich nicht nur in der theoretischen Informatik der Gegenwart einer regen Forschungstätigkeit, sondern dürfte philologisch Gebildeten nicht ganz unbekannt vorkommen, stellt es doch eine recht alltägliche Problematik bei der Herstellung historisch-kritischer Editionen dar, wo ebenso zwischen verschiedenen Varianten eines Texts zu differenzieren und anschließend eine Entscheidung zu fällen ist, welcher Variante der Vorzug zu geben sei.

```

--- Original.txt      2018-03-01 21:09:05.000000000 +0100
+++ OriKopie.txt      2018-03-01 21:09:09.000000000 +0100
@@ -8,7 +8,7 @@
    gegeben("Paris",1781-08-04)); // 3. Brief

    readLetter("Vicomte de Valmont", "Marquise de Merteuil",
-       gegeben("Schloß Cormatin",1781-08-05)); // 4. Brief
+       gegeben("Schloß Chambord",1781-08-05)); // 4. Brief

    readLetter("Marquise de Merteuil", "Vicomte de Valmont",
    gegeben("Paris",1781-08-07)); // 5. Brief

```

Abbildung 2: *diff* von Original und Kopie

Ich möchte nun auf die drei medialen Praktiken *branch*, *diff*, *merge* etwas genauer eingehen, indem ich nun meinerseits in der Zeit etwas zurückgehe, um die Verfahren, wie sie in informatischen Kollaborationen derzeit weltweit Anwendung finden, auf ihre eigene Geschichte hin zu befragen. Denn ein besonderer Vorzug von Versionskontrollsystemen besteht darin, dass ein solches System stets reversibel in der Zeit bleibt, das heisst, man kann nahezu mühelos zwischen unterschiedlichen Zeitpunkten der Codeentwicklung wandern, die Zeitachse also bei Bedarf wie einen Schieber zurück bewegen, um jegliche Änderung am Code wieder ungeschehen zu machen. Hier zeigt sich auch schon eine wichtige Differenz: Weder in der Historiographie von Software noch in der von Kulturtechniken wie dem Kopieren besteht (bedauerlicherweise) diese Möglichkeit einer Rückkehr zum *status quo ante* in dieser Form, weil wir uns stets im Hier und Jetzt befinden. Es sei denn, wir bewegen uns immersiv durch ein spezifisches Medium in andere Welten, womit ganz andere Effekte möglich werden. Was nach *Science Fiction* klingt, erweist sich ganz buchstäblich als der zweite Terminus, denn dieses Medium der Zeitreise ist die Fiktion.



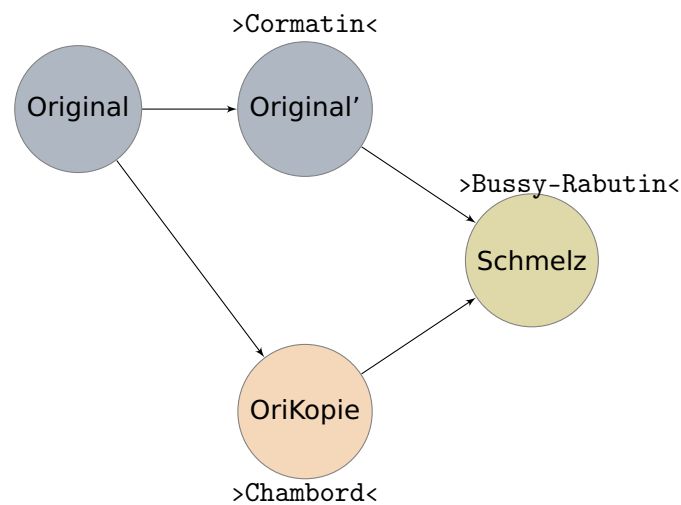


Abbildung 3: Neue Variante der Varianten

## 1 Verzweigen

Der unscheinbare Befehl *branch* bewirkt nichts weniger, als mit einem einzigen schlichten Kopiervorgang ein Paralleluniversum zu erzeugen. Der gesamte Kosmos des Softwareprojekts [*Gefährliche Liebschaften*] wird mit all seinen Routinen, Nischen, Fehlern, Kommentaren, Besonderheiten und Unzulänglichkeiten leichterhand dupliziert, um sodann lediglich in einem kleinen Detail verändert werden zu können. Der Zeitpunkt der Befehlserteilung *branch* markiert den Bifurkationspunkt, ab dem sich die Welten teilen, um fortan zwei parallele Weltläufe mit zwei »verschiedenen Zukünften«<sup>3</sup> zu generieren. Was dem einen als ein »wirrer Haufen widersprüchlicher Entwürfe« erscheint, ist für den anderen ein wohlgeordnetes »Labyrinth aus Symbolen«.<sup>4</sup> Oder noch genauer: ein »Labyrinth aus Zeit«.<sup>5</sup> Denn diese Struktur aus parallelen Welten mit ihrer unbegrenzten Möglichkeit zur Kontingenz »erschafft so verschiedene Zukünfte, verschiedene Zeiten, die ebenfalls auswuchern und sich verzweigen.«<sup>6</sup>

Was manche von Euch vermutlich längst bemerkt haben, obwohl ich die hier notierten Operatoren, also An- und Ausführungszeichen nicht mitlas, ist der Umstand, dass ich meinen Text selbst gerade verzweigt habe, und zwar indem ich einen Teil der charakterisierenden Beschreibung einfach hineinkopiert habe, indem ich aus einer Erzählung von Jorge Luis Borges aus dem Jahre 1941 zitierte. Wie in dem Briefroman von Choderlos de Laclos gibt es in dieser Erzählung eine Rahmenhandlung, in der ein fiktiver Herausgeber dem Leser von einem unverhofft gefundenen Text-Fragment

3. borges:1941.

4. borges:1941.

5. borges:1941.

6. borges:1941.

berichtet, das eine zuvor noch rätselhafte Kriegshandlung erhellt. In diesem Fragment wird berichtet, wie der Ich-Erzähler, ein chinesischer Spion, der im Ersten Weltkrieg in England für die Deutschen kundschaftet, den britischen Sinologen – nicht Gleb, sondern: – Stephen Albert aufsucht. Bei diesem Besuch begegnet der Chineser einer besonderen Struktur, und zwar dem von einem seiner Vorfahren entworfenen »Garten der Pfade, die sich verzweigen«, – so der Titel der Erzählung. Dieser labyrinthische Garten erstreckt sich jedoch nicht im Raum, sondern in der Zeit, insofern er aus einem in zahlreichen Versionen durchgespielten Roman besteht, der – wie in Leibniz Theodizee<sup>7</sup> – alle möglichen Begebenheiten in parallelisierten Varianten enthält. Der Autor dieses Romans »glaubte an unendliche Zeitreihen, an ein wachsendes, schwindelerregendes Netz auseinander- und zueinanderstrebender und paralleler Zeiten. Dieses Webmuster aus Zeiten, die sich einander nähern, sich verzweigen, sich scheiden oder einander jahrhundertlang ignorieren, umfaßt alle Möglichkeiten.«<sup>8</sup> Wie sich diese Problematik von Unendlichkeit materiell bewerkstelligen lässt, wird in der Erzählung freilich ebenso reflektiert, nämlich entweder zyklisch, so wie es Vladimir Nabokov mit seinem Karteikarten-Roman *Pale Fire* zwei Jahrzehnte später vorführt, oder rekursiv, wie Scheherazade, die an einer bestimmten Stelle in *Tausendundeiner Nacht* aus einer bestimmten Stelle von *Tausendundeiner Nacht* vorliest. Es ist ein »Labyrinth, das die Vergangenheit umfaßte und die Zukunft [...] Zum Beispiel kommen Sie in dieses Haus, aber in einer der möglichen Vergangenheiten sind Sie mein Feind gewesen, in einer anderen mein Freund.«<sup>9</sup> Und wie die Geschichte dann weiter zeigt, wird er auch beides *zugleich* gewesen sein. . .

Es ist eigentlich kaum nötig zu erwähnen, dass diese Charakteristik des unendlichen Romans, die Borges entwirft, eine ziemlich präzise strukturelle Beschreibung dessen ist, was eine Software-Versionsverwaltung bereitstellt: »ein wachsendes, schwindelerregendes Netz auseinander- und zueinanderstrebender und paralleler Zeiten«. Die Software-Versionsverwaltung agiert hier als ein System, das sich als ziemlich fundamental erweist, sorgt es doch dafür, die temporalen Unterschiede zu verwalten, die Vergangenheit als Archiv genau zu dokumentieren, um aus diesem Archiv heraus die Zukunft zu bahnen. Die Versionsverwaltung gerät zu nichts weniger als der Herrin der Zeit.

Bevor ich nun von der medialen Praktik des Verzweigens meinerseits wieder verzweige auf die Unterschiedsbestimmung mit *diff*, sei noch ein

---

7. Die Konstruktion gleicht in gewisser Weise der Konstellation, die Leibniz in seiner Theodizee-Problematik durchspielt: Die göttliche Ordnung kennt zahlreiche Welten, die nebeneinander bestehen, sich gleichen, aber doch in einigen signifikanten Details unterscheiden. Allerdings sind im Fall der Software keine Qualitätskriterien wie gut und böse ausschlaggebend, sondern eher der Zuspruch und die Nutzung durch andere Entwickler. Zudem führt die Parallelität der beiden Code-Ordnungen (um nicht Kosmoi zu schreiben) vor allem die Kontingenz vor Augen, dass jede artifizielle Welt auch anders sein könnte.

8. **borges:1941.**      9. **borges:1941.**

kleiner Unterzweig eingebunden, der so etwas wie die eminente Übertragungsfunktion aller drei Verfahren darstellt. Denn weder *branch* noch *diff* oder *merge* kommt ohne einen Vorgang aus, der die Daten von A nach B schafft, prüft, validiert, transferiert und dupliziert. Es folgt demnach eine kurze Verzweigung zum Kopieren...

## 2 Kopieren

Ich möchte im Folgenden den Fokus auf eine ebenso hochprofessionalisierte wie immer schon international arbeitende Institution lenken, wo das Vervielfältigen in Serie, der Akt des Kopierens, des Filterns und ggf. noch des Verschlüsseln zur exklusiven Aufgabe zählte. Die Rede ist von einer Institution, die unter wechselnden Namen wie ›geheimbe Zyffer Weeßen‹, ›Zyffer Scretariat‹, ›Kabinets-Secretariat‹, ›Visitations- und Interceptions-Geschäft‹, ›Geheime Kabinets-Kanzlei‹ oder auf ihren französischen Ursprung unter Ludwig XIV. rekurrierend als *cabinet noir* oder schlicht als ›Schwarzes Kabinett‹ bezeichnet wird. Dieser auch als ›Brief-Inquisition‹ bezeichneten Abteilung im Umfeld eines weltlichen Herrschers unterstand es, den gesamten Postverkehr eines Landes, insbesondere in der Residenzstadt zu überwachen, die wichtigen Briefe nicht nur abzufangen, sondern unbemerkt zu entsiegeln, zu öffnen, zu durchmustern, zu lesen, sie ggf. zu entschlüsseln, zu kopieren, zu registrieren, zu verschliessen, endlich sie mit einem gefälschten Siegel zu versehen, um sie daraufhin wieder dem ursprünglich beabsichtigten Postlauf zu übergeben.

Insbesondere in Wien, der Stadt des Kaisers, legt man viel Wert auf einen geräuschlosen Ablauf im Hintergrund des weitverzweigten Postwesens, das nicht zuletzt von der traditionellen Nähe der Habsburger zur Familie Thurn und Taxis bestimmt wird. In unmittelbarer Nähe zur Macht, vis-à-vis zur Wiener Hofburg, residiert diese Reichszentrale Intelligenz-Agentur, um ihrer eigenen Auffassung von Aufklärung nachzugehen:

Abends Schlag 7 Uhr schloß sich die Postanstalt und die Briefwagen schienen abzufahren. Sie begaben sich aber in einen Hof des kaiserlichen Palastes, woselbst schwere Thore sich sogleich hinter ihnen schlossen. Dort befand sich das Schwarze Kabinett, die Stallburg.

Da öffnete man die Briefbeutel, sortirte die Briefe und legte diejenigen bei Seite, welche von Gesandten, Banquiers und einflußreichen Personen kamen. Der Briefwechsel mit dem Auslande zog meist ganz besondere Aufmerksamkeit auf sich. Die Siegel wurden abgelöst, die wichtigsten Stellen kopirt und die Briefe mit teuflischer Geschicklichkeit wieder verschlossen.<sup>10</sup>

---

10. **koenig:1875.**

Wenn selbst die wichtigsten Sendschreiben nicht lange verweilen dürfen, um noch in derselben Nacht auf den Weg ihrer eigentlichen Bestimmung gebracht zu werden, ist stets Eile geboten. Zwischen 80-100 Briefe schafft man täglich zu durchmustern bzw. zu perlustrieren – wie es im Fachjargon heisst. Nach einer ersten Sichtung von Adressat und Absender, also einer Registrierung anhand der Meta-Daten, erfolgt bei Schreiben, die weitergehendes Interesse verheissen, eine genauere Autopsie des derart entwendeten Briefes, der »mittels einer sehr dünndochtigen brennenden Kerze mit ›unruhiger‹ Hand aufgelassen und geöffnet [wurde]. Der Manipulant merkte sich schnell die im Kuvert liegenden Bestandteile, die Lage derselben und übergab das Briefpaket dem Subdirektor, der den Brief durchlas und entweder den ganzen Inhalt oder ihn auszugsweise kopieren liess.«<sup>11</sup> Nach einer ersten Übersicht der einzelnen Programmbestandteile, also einer Sichtung ihrer Lage, wird der Code weitergereicht, um zweitens, noch ggf. entschlüsselt und dann interpretiert zu werden, bevor man ihn drittens, erneut arbeitsteilig zu kopieren sich anschickt. – Ich hebe das noch einmal eigens hervor, weil diese drei Schritte ebenso konstitutiv sind für den Vorgang des *branching*, dem sie notwendigerweise vorausgehen. Der eigentliche Kopiervorgang erfolgt sodann unter Einsatz einer medientechnisch verfeinerten *ars dictaminis*:

Die Offiziale waren in der Regel Schnellschreiber. Hin und wieder gab es auch ›short hand-Schreiber‹. Man diktierte, um Zeit zu gewinnen. Zwei Offiziale nahmen einen Bogen und diktieren zwei Schnellschreibern, ja sogar vier Offiziale diktierten zugleich aus einem Bogen auf eine so geschickte Weise vier Kollegen, daß die Schreibenden nicht irre werden konnten. Auf diese Weise konnte ein Bogen in wenigen Minuten abgeschrieben werden. Im Notfalle kopierte das ganze Personal ohne Unterschied, Hofrat und Subdirektor mitinbegriffen.<sup>12</sup>

Der Kabinettsdirektor überprüft anschliessend diese derart im beschleunigten Multitasking oder Parallelprocessing gewonnenen Ergebnisse, filtert sie nach der jeweiligen politischen Interessenslage und reicht sie weiter direkt zum Kaiser bzw. zur Polizei.<sup>13</sup> Einen halben Briefbogen pro Minute kennzeichnet eine Datendurchsatzrate, die nicht so viel geringer bleibt als die einer *floppy disk* 200 Jahre später, zumal deren Weiterentwicklung in Form der Festplatte auch mit verteilten Rollen, das heisst mehreren Schreib-Lese-Köpfen zu arbeiten pflegt. Eine allfällige Verschlüsselung beansprucht um 1780 ebenfalls nur etwas mehr Zeit als um 1980. Es kann daher nicht verwundern, wenn den Schwarzen Kabinetten schon im 19. Jahrhundert der Nimbus einer modernen, mit der neusten Medientechnik ihrer Zeit experimentierenden Intelligenzagentur konstatiert wird. »Sie gleichen weniger Postämtern, als Laboratorien.«<sup>14</sup>

11. **stix:1937.**    12. **stix:1937.**    13. **stix:1937.**    14. **koenig:1875.**

Eine der entscheidenden Fähigkeiten, die in diesen Laboratorien stets geübt und weiter entwickelt werden, besteht in der Nachahmung der jeweiligen Handschriften. Kopieren bedeutet nämlich nicht nur, den Inhalt buchstabengetreu von einem Blatt auf das andere zu übertragen, sondern ebenso, sich des Stils, der Eigenheiten, der inneren wie der äusseren Form des Anderen im Brief – und nicht selten auch über den Brief hinaus – anzuverwandeln. »Man öffnete die Briefe, schrieb sie ab und unterschob perfide Schreiben, in denen Handschrift, Schreibweise und Überschrift des Absenders mit wunderbarer Kunst nachgeahmt war«, fasst Emil König in seiner Streitschrift gegen die Verletzung des Briefgeheimnisses von 1875 diese Fähigkeit lakonisch zusammen.<sup>15</sup> Den Kopisten selbst schreibt König dabei eine derart exzessive mimetische Kraft zu, dass es nicht selten psychopathologische Züge annehme: »Nicht genug, daß sie die Briefe mit einer ganz erstaunlichen Gewandtheit öffneten und wieder versiegelten, ahmten sie auch die Schriftzüge nach, schrieben falsche Briefe, gaben falsche Rathschläge und betrogen Absender und Empfänger auf das Schändlichste. Ihre Arbeit erforderte übrigens eine so große Anspannung des Geistes, so viel Sorgfalt und Geschwindigkeit, dass mehrere dadurch den Verstand verloren.«<sup>16</sup>

Keine Kopie ist authentisch oder fehlerfrei, weder in technischen Medien wie der Photographie, wo sich das Verfahren selbst ins Bild einschreibt, noch in mimetischen Prozessen wie dem Kopieren eines bestimmten Habitus oder einer Handschrift. Erst mit der Möglichkeit digitaler Vervielfältigung wird Kopieren zu einem Akt, der das Artefakt so dupliziert, dass ohne Meta-Daten wie Time-Stamp, Entstehungsdatum, Zeitpunkt letzter Änderung etc. kein Kriterium mehr gegeben ist, um Original und Nachbildung zu unterscheiden. Schon aus diesem Grund ist es wichtig, bei der Versionskontrolle eine Fehlerkorrektur bzw. Validierungsinstanz zu haben, die im Zweifelsfall die Unterschiede zwischen Original und Kopie zu finden verspricht. Und damit komme ich endlich zur zweiten Praktik, dem *diff*-Befehl, den ich vergleichsweise kurz halten werde.

### 3 Abweichen

Es gibt verschiedene Formen der textuellen Abweichung im Kontext kollektiver Autorschaft und Versionskontrolle. So erscheint es manchmal erforderlich, in einem gemeinschaftlich verfassten Text eine Differenz zu markieren.<sup>17</sup> Oder es gilt, einen einfachen Ausgangssatz in verschiedensten Stilausprägungen zu variieren, wie es Raymond Queneau 1947 in seinen 100 Varianten der *Exercices de style* vorgeführt hat, um die Kontingenz der unterschiedlichen Stile und rhetorischen Figuren vorzuführen.

---

15. **koenig:1875.** 16. **koenig:1875.** 17. **fuchs:1998a.**

<pre> gegeben("Paris",1781-08-03)); // 1. Brief  readLetter("Marquise de Merteuil", "Vicomte de Valmont",     gegeben("Paris",1781-08-04)); // 2. Brief  readLetter("Cécile Volanges", "Sophie Carnay",     gegeben("Paris",1781-08-04)); // 3. Brief  readLetter("Vicomte de Valmont", "Marquise de Merteuil",     gegeben("Schloß Cormatin",1781-08-05)); // 4. Brief </pre>
<pre> readLetter("Marquise de Merteuil", "Vicomte de Valmont",     gegeben("Paris",1781-08-07)); // 5. Brief </pre>
<p>Dieser Teil ist nur im Text Original.txt enthalten:</p> <p>Die kürzeste Kurzgeschichte der Literaturgeschichte lautet: "Als er aufwachte, war der Dinosaurier immer noch da..."</p>
<p>Dieser Teil ist wieder in beiden Texten vorhanden...</p>
<p>status: 3 differences</p>

Abbildung 4: *diff* von Original und Kopie vor dem *merge*

Wo wäre in der langen Geschichte kollektiver Autorschaft der historische Vorläufer zum *diff* zu verorten? Welche Instanz sorgt sich um etwaige Satz- oder Tippfehler, Zeilen-Unterschiede, Kopierfehler, unmerkliche, wenn nicht infinitesimale Details in der Wort(dar-)stellung? Kurzum, was ist das klassische Pendant zum *diff* und seiner Fixierung der Deltas? Ebenso kurz gesagt: Der Herausgeber oder Bearbeiter einer Edition, derjenige also, der für die Sicherung des Textes, für die Entscheidung, dieser und nicht der anderen Variante den Vorzug zu geben, verantwortlich zeichnet, wobei er freilich – etwa bei historisch-kritischen Editionen – die Kontingenz der Varianten ebenfalls zur Darstellung zu bringen hat. Im *diff* kondensiert – oder mit Blick auf den letzten Abschnitt: schmilzt – also eine lange Theoriegeschichte der Philologie und Editionswissenschaft.

## 4 Verschmelzen

Die Zeichenzahlvorgabe reicht an dieser Stelle nicht mehr aus, nun noch die dritte Praktik *in extenso* vorzuführen, die grundlegend für die kollektive Autorschaft in der Softwareentwicklung und ihrer Versionsverwaltung bleibt [Abb. 3 ●].

Ich möchte daher nur kurz skizzieren, was ich detaillierter darzulegen wäre: Und zwar hätte ich Ihnen zu zeigen versucht, wie die Parallelität verschiedener Zeiten, die auswuchernden Zweige und Gabelungen in einzelnen Fällen wieder zusammengeführt werden, um dank der jeweils erfolgten Umwege einen neuen, dritten, synthetisierten Wissensstand zu bündeln. Vorgeführt hätte ich dies gerne – anders als bei Borges und seinem unendlichen Buch als Labyrinth – zwar ebenso mit einer ›prinzipiellen Unendlichkeit‹ (Luhmann), allerdings in anderer Form, und zwar anhand eines kollektiven Buchs, das seine eigene Abschaffung als Buch vollführt: Ein Buch, das nichts als Bücher verzeichnet, ein Buch, das von vielen Autoren geschrieben ist und noch viel mehr Autoren vereint, weil es deren Metadaten listet. Ein Buch, das sich in seiner Form nur noch als Buch tarnt [●], obwohl es seine Bestandteile längst schon dissolviert oder herausgelöst hat. Ein Buch über Bücher, das aus nichts als frei verschiebbaren Zetteln besteht. Mit einem Wort, ein Katalog, und zwar nicht irgendeiner, sondern der erste Zettelkatalog der Bibliotheksgeschichte. Dass diese Arbeit an einem derart weit verzweigenden Projekt einer dezidierten Arbeitsteilung unterliegt, mag kaum überraschen. Umso konsequenter erscheint die Kodifizierung dieser Tätigkeit, die kaum zufällig schon in der Wiener Hofbibliothek um 1780 eine algorithmische Struktur annimmt. Vorzuführen gewesen wäre also der *merge*-Algorithmus oder die sog. Instruktionen, mit dem aus vielen Büchern ein einziges wird, mit dem viele Autoren zu einer Struktur zusammengebunden werden, die wiederum neue Autoren aufgreifen und produzieren soll. Es wäre zu zeigen, wie aus dieser informationellen Vereinzelung, Fragmentarisierung, Beweglichkeit, Atomisierung der Informationsbausteine wieder ein einziger Strang wird, ein Faden oder Pfad, der mit seinen volatilen Elementen seinerseits und jederzeit neue Verzweigungen oder auch Weiterführungen auf demselben Weg erlaubt.

Aber, alles das jetzt nicht. Sondern statt eines neuen Codes aus dem späten 18. Jahrhundert nur noch, in sechs Minuten, – eine Coda.

## 5 Coda: Assistenzsysteme, Quellcodekritik und Auto(r)Korrektur

Was ich bis jetzt ansatzweise versucht habe zu skizzieren, liesse sich unter dem heuristischen Begriff einer ›Quellcodekritik‹ fassen, ein Kofferwort aus Quellcode und der guten alten historiographisch-hilfswissenschaftlichen

Quellenkritik. Dabei geht es mir in diesem Zuschnitt einer Methodik allerdings nicht so sehr darum, Algorithmen zu kommentieren und den Code einer Software auf seine Stimmigkeit, Effektivität oder Eleganz hin zu lesen – das auch, aber eher mit nachgeordneter Priorität. Vielmehr geht es darum, jenseits des Inhaltlichen und der Funktionalität einer Software die Materialitäten der Kommunikation auch im Virtuellen, in diesem Fall das global verteilte, kollektive Softwareentwickeln – ganz im Sinne einer klassischen Quellenkritik – auf seine medialen Praktiken und archivalischen Strukturen hin zu befragen und um die Perspektive einer historischen Genese zu erweitern. Denn keine Denkfigur oder Praktik innerhalb der Informatik im Allgemeinen und der Softwareentwicklung im Besonderen kommt ohne eine entsprechende, zum Teil weit zurückreichende Genealogie aus. Der Umstand, dass man beim Programmieren immerzu auf *Bibliotheken* zugreift, auch wenn es sich dabei um \*.jar, \*.zip, \*.dylib, \*.lib- oder sonstige Programmbibliotheken handelt, bedarf keines weiteren Kommentars. Dass die weitestgehend geschichtsvergessene *computer science* auf die Historizität von Algorithmen, Programmiersprachen, Entwicklungsumgebungen – jenseits von Kompatibilitätsfragen – für gewöhnlich wenig Aufmerksamkeit richtet, mag wohl unabänderlich sein. Dass aber die kulturwissenschaftlich angeleiteten *software studies* hier noch lohnende Desiderate finden können, liegt auf der Hand. Ich möchte daher abschliessend skizzieren, wie eine solche Methodik und Richtung potentieller Forschungsfragen aussehen könnte, um einen Wissenstransfer aus der historischen Analyse in die praktische Codeentwicklung zu leisten.

*branch*, *copy*, *diff*, *merge* und einige andere Befehle zählen zu den eminenten Funktionen verteilter Code-Autorschaft in der Softwareentwicklung. Wie ich versucht habe zu zeigen, basieren diese Funktionen auf Praktiken, die sich zum einen in der Literaturgeschichte und ihrem Zusammenspiel aus experimenteller Autorschaft und Editionstätigkeit gründen (Borges und *branch*), zum zweiten in der Geschichte der Telekommunikation und im Postverkehr (Schwarze Kabinette, *copy*, *decode*, *diff*) und schließlich auch in den Verwaltungspraktiken des Wissens, konkret in der Katalogarbeit der Aufklärung (Wiener Hofbibliothek und ihre Zettelkatalog, *merge*). Diese historische Konstellierung mag – wenngleich nur schlaglichtartig oder exemplarisch – vor Augen führen, aus welchen Bestandteilen die informatische Versionsverwaltung verfertigt ist, ohne es zu wissen. Die Quellcodekritik zielt aber nicht allein darauf, die aktuelle Funktionsweise der *computer science* historisch nach hinten zu verlängern, sondern ebenso umgekehrt, aus und vor allem *in der Tiefe der Geschichte* nach Denkfiguren und Funktionsweisen, nach medialen Praktiken und historischen Tätigkeiten, nach Fragestellungen und Problemlösungen zu suchen, um diese Erkenntnisse in die Softwareentwicklung hineinzutragen. Wie könnte das aussehen?

Die Reichweite eines Befehls wie *diff* mag für informatische Zwecke



begrenzt und aus historisch-kritischer Perspektive für einen Editor zudem keineswegs neu sein; was aber wäre, wenn es einen Befehl gäbe, der Differenzen auch auf einer inhaltlichen Ebene vorschlagen könnte? Also einen Befehl, der sich an Autoren im Schreibprozeß richtet und sich dabei weniger an Herausgeber-Funktionen orientierte als vielmehr an den Leistungen eines Verlegers oder Lektors, der einem um den guten Ausdruck bemühten Autor mehr anbietet als lediglich *diff*-gleich nur Abweichungen zu markieren? Was wäre, wenn der Algorithmus, statt bloss die unmerklichen bis infinitesimalen Deltas zu verzeichnen, selbst Kreatives leistete? Wenn beim Schreiben in den Officeprogrammen auf Befehl die Routine eines Lektors abzurufen wäre, der die subtilen Stiländerungen souverän erkennt und seinerseits durch feine Vorschläge variieren kann? Gesucht wäre also so etwas wie ein schlauer Diener beim Schreiben, ein Assistenzsystem für die treffende Formulierung, ein Lektorats-Algorithmus, der nicht nur Goethe und Schiller unterscheiden kann in ihrer jeweiligen Stilistik, sondern auch noch Goethe alternative Vorschläge à la Schiller unterbreitet und vice versa, wenn Schiller stockt beim Schreiben die Vorschläge à la Goethen einspeist.

Ein solcher Algorithmus würde folgende Arbeitsschritte umfassen, indem er einerseits eine informatische Stilanalyse verfolgte, andererseits aber auch eine softwareseitige Stilgenese anböte, um überhaupt neue Vorschläge unterbreiten zu können. Mit anderen Worten, dieses Assistenzsystem arbeitet mit den Lehren der Vergangenheit, um künftigen Texten den Weg vorzuspüren. Denn »Assistenzsysteme dienen den Nutzern zur Unterstützung in bestimmten Situationen oder bei bestimmten Handlungen. Die Voraussetzung dafür ist eine Analyse der gegenwärtigen Situation und gegebenenfalls darauf aufbauend eine Vorhersage der zukünftigen Situation. Die Interaktion sollte sich dem natürlichen Handlungsablauf des Menschen anpassen und die Ausgabe sollte komprimiert sein, um den Nutzer nicht zu überlasten.«<sup>18</sup> Kaum notwendig zu erwähnen, dass diese informatische Definition als Gegenüber des Menschen die Maschine setzt, die medienhistorische Perspektivierung hier jedoch ganz allgemein ein Medium identifiziert, das menschlicher wie nicht-menschlicher Provenienz sein kann. Wie sähe eine Modellierung eines solchen Stil-Befehls aus?

- Einlesen eines Beispieltexsts, eines Text-Korpus, eines ganzen Werks zur Stilanalyse
- Diese Art der stilistischen Mimesis oder Originalkopie speist sich aus der *copia verborum*, aus der Fülle der Wörter, also aus einer rhetorischen Funktion, die ohne grösseren Aufwand computertechnisch mit Hilfe von Markov-Ketten, also einer statistischen Auswertung der

---

18. [dbis.informatik.uni-rostock.de/forschung/schwerpunkte/assistenzenzsysteme/](http://dbis.informatik.uni-rostock.de/forschung/schwerpunkte/assistenzenzsysteme/)

Übergangswahrscheinlichkeiten von Worten von Worten von Worten, nachgebildet werden kann.

- Nach einer bestimmten Anlernphase, innerhalb derer sich der Algorithmus den Stil eines bestimmten Textes oder gar Autors aneignet, würde ein Repositorium geschaffen, mit dessen Hilfe beim Formulieren Vorschläge unterbreitet werden können.

Was also ein solcher Algorithmus auf Basis einer Markov-Ketten-Analyse liefern würde, wäre eine stilistische Anverwandlung einer bestimmten Autorschaft, je nachdem, was man ihm einfüttert, ergeben sich typische Formulierungshilfe: Beim Einlesen des Götz von Berlichingen gäb's einen mittelalterlichen Alltagssound, wo auch mal derbe Worte fallen. Beim Einlesen von Kafkas Prozeß gäbe es kristallklare Prosa als Formulierungshilfe, und beim Einlesen des Autors von *Sein und Zeit* gäbe es einige nur bedingt hilfreiche, weil selbstbezügliche Formulierungsvorschläge, weil die ganze Welt plötzlich zu welten beginnt.

Mit einer solchen Anordnung von ›mimetischen Algorithmen‹ könnte es demnach gelingen, einerseits Fragen der *software studies* ins 18. Jahrhundert zu tragen, also auf die ›alten‹ Praktiken der Exzerpt-, Fragment- und Informationsschnipsel-Verarbeitung im Kontext einer kollektiven Autorschaft zu beziehen. Und umgekehrt eröffnet sich damit ein Weg, durch eine Analyse der Instruktionen und Verfahren, mit denen kollaborative Autorschaft in den unterschiedlichsten Formen und Situationen historisch zur Ausführung gelangten, die komplexen, distribuierten, wolkenbasierten Verfahren verteilter Autorschaft der Gegenwart nicht nur zu verstehen, sondern – *historia est magistra codicum* – aus der Tiefe der Geschichte heraus weiter zu entwickeln, um so zu einer wechselseitigen Erhellung von Code-Entwicklung und einer historischen Forschung zu gelangen. Doch dazu ist es einstweilen notwendig, noch andere lose Enden aufzunehmen, um sie weniger zu verschmelzen als – und damit leite ich schon über zu Helgas Referat – zu verflechten.