›branch‹, ›diff‹, ›merge‹

Versionskontrolle und Quellcodekritik

Markus Krajewski, Universität Basel,

local version 0.75:2019-02-27,
github.com/nachsommer/VersionsKontrolle: 588584-dirty

erscheint in: Jörg Paulus, Andrea Hübener (Hrsg.),
Abschrift, Ablichtung, CC & Vice Versa, tba, 2019

Inhaltsverzeichnis

Zusammenfassung

Der Beitrag wirft einen Blick hinter die Kulissen, wie grossangelegte kollektive Schreibprojekte medientechnisch organisiert sind und welchen Befehlssätzen und Operationsketten sie gehorchen. Im Fokus stehen dabei einige historische Szenarien, in denen sich die kulturtechnische Funktionsweise verteilter Autorschaft präfiguriert. Das Ziel dieser medien- und kulturhistorischen Perspektivierung liegt darin, der Geschichtsvergessenheit der *software studies* ein wenig entgegenzuarbeiten, um die gegenwärtige Praktik kollektiver Autorschaft in der Softwareentwicklung zu ihren historischen Wurzeln und Entwicklungslinien zurück zu verfolgen, nicht zuletzt geleitet von dem Anspruch, die informatischen Praktiken nicht einfach als immer schon gegeben hinzunehmen, sondern sie selbst durch ihre vorgängigen Entwicklungslinien im vorelektronischen Zeitalter zu ergänzen. Drei exemplarischen Geschichten oder historischen Szenarien gilt dabei eine besondere Aufmerksamkeit: Sie ereignen sich nahezu synchron um 1780, zum einen im Frankreich des Ancien Regime, zum zweiten

in Wien, der Hauptstadt des Heiligen Römischen Reichs Deutscher Nation, und zum dritten im Herzogtum Sachsen-Weimar-Eisenach, daneben aber auch in London 1916 und anderenorts. Dieses Verfahren, informatische Strukturen der Gegenwart mit ihren kulturhistorischen Vorläufern und Wegbereitern zu erläutern, wird am Ende dieses Beitrags gebündelt durch das Konzept der Quellcodekritik, das es erlaubt, die in Algorithmen eingekapselten Geschichten ihrerseits lesbar und kritisch nachvollziehbar zu machen.

Vorbemerkung

Dieser Text, der sich mit der Geschichte und den medialen Praktiken von Versionierungssystemen befasst, existiert selbst in drei Versionen, die sich jedoch – ganz wie die historisch zurückzuverfolgenden Funktionen der Versionskontrolle bei der kollaborativen Textproduktion – signifikant voneinander unterscheiden.

Zwar folgen die drei Versionen einem gemeinsamen Argumentationsgang, sie weichen jedoch in einigen Passagen und Details deutlich voneinander ab, insofern erst in der dritten Fassung (commit 3fsd57) das Konzept der Assistenzsysteme vorgeschlagen wird, während in der vorliegenden, ersten Fassung (commit 6g8wkit9) der Fokus auf den Verfahren und dem Konzept der Quellcodekritik liegt, und in der zweiten Fassung (commit 9gkfur5) die Passagen über die kollektive Autorschaft, insbesondere beim Vorgang des Verschmelzens, grösseren Raum einnehmen. Eine Gesamtsicht der Versionsänderungen lässt sich mit Blick auf die Unterschiede in den jeweiligen Ausgangsdateien (im LaTeX-Format) nachvollziehen. Zur besseren Lesbarkeit liegen die Texte in ihren drei Varianten nicht nur auf einer linearen Zeitachse angeordnet, sondern – wie bei Borges – parallel in drei Fassungen auf GitHub sowohl als LaTeX- als auch als pdf-Dateien vor.

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 Verwendung, konkret bei der sogenannten *version control*. 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 unterschiedlichen Orten gleichzeitig an demselben Code arbeiten, dessen Änderungen demzufolge zeichengenau protokolliert werden und nachvollziehbar bleiben müssen.[1] Das bekannteste System

1. Vgl. Yuill 2008.

dieser Art dürfte derzeit die von Linus Torvalds initiierte Platform *github* sein, auf der unter github.com/nachsommer/VersionsKontrolle auch eine digitale Version dieses Texts zu finden ist.

Das Ziel der Codeentwicklung 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 Code der Maschinensprache übersetzt. Auch bei diesem Übersetzungsprozess wird jedes Zeichen, jeder Befehl, jede Schleife, jede Datenstruktur sequentiell eingelesen, validiert und interpretiert. Man muss sich den *compiler* als einen Meister des *close reading* vorstellen.[2] 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 in entsprechenden Repositorien 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:

```
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
```

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‹

2. Hier wäre noch auf die kodifizierende Funktion, das Schliessen des Codes, hinzuweisen, die ja auch vom Compiler vorgenommen wird, vgl. Krajewski und Vismann 2009.

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 Postillon seine Briefe zustellen konnte, was wiederum 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 bleibt ungelöst und der Programmcode wird an dieser Stelle kurzerhand verzweigt, das heisst mit Hilfe des Befehls *branch* gelingt es, den Code zu duplizieren, um beide Varianten parallel zueinander existieren zu lassen (Abb. 1). 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.

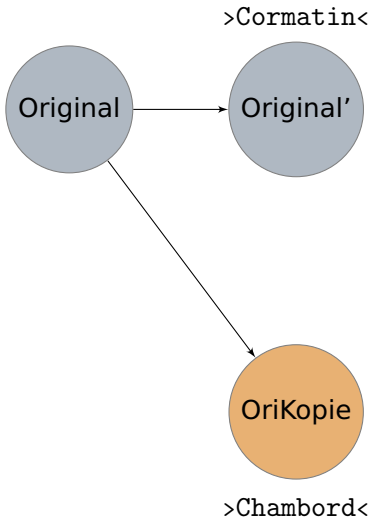

Abbildung 1: Varianten einer Entwicklung

Beide Stränge sind nach wie vor für alle Entwickler sichtbar und nahezu identisch, bis auf die kleine Abweichung von Egmont, der seinen *source code* gegenüber dem zwischenzeitlich womöglich ebenfalls weiter entwickeltem Original' von Richard 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 (Abb. 2).

diff macht also den kleinen Unterschied sichtbar. Das Programm hebt die Abweichungen zweier in weiten Teilen identischen Dokumente hervor,

```
--- 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

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 *différance* 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 *différance* 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 ganz unterschiedliche Programme geworden sein werden. Doch die auseinanderstrebende Bewegung der beiden Teile wird in diesem (fiktiven) 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 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 Gebildeteten 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.

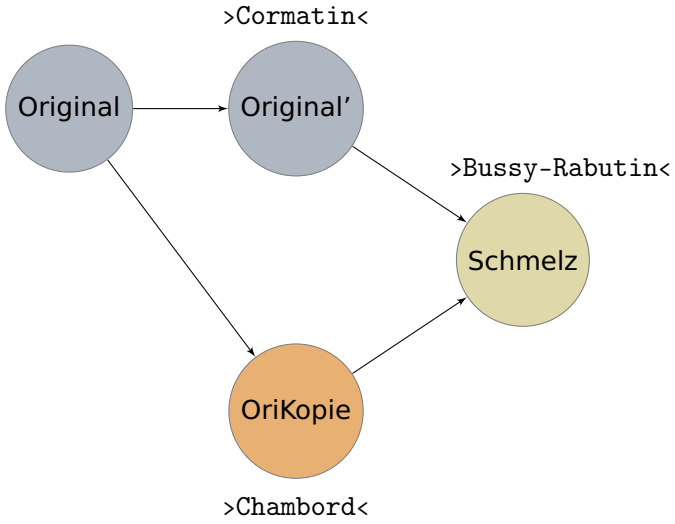

Abbildung 3: Neue Variante der Varianten

Auf die drei medialen Praktiken *branch*, *diff*, *merge* sei nun etwas genauer eingegangen, 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 Geschichtsschreibung von Kulturtechniken wie dem Kopieren besteht (bedauerlicherweise) die Möglichkeit einer Rückkehr zum *status quo ante*. Es sei denn, man bewegt sich immersiv durch ein spezifisches Medium in andere Welten. Und dieses Medium ist die Fiktion.

1 Verzweigen

Der unscheinbare Befehl *branch* bewirkt nichts weniger, als mit einem einzigen schlichten Kopiervorgang ein Paralleluniversum zu erzeugen. Der

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

Die Erzählung von Jorge Luis Borges von 1941, aus der die Zitate im vorangehenden Absatz stammen, entwickelt eine Struktur, die jener eigenartigen textuellen Parallelwelt entspricht, die durch einen *branch*-Befehl erzeugt wird. 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 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 Stephen Albert aufsucht. Bei diesem Besuch begegnet der Chinese 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[7] – 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 jahrhundertelang ignorieren, umfaßt alle Möglichkeiten.«[8] 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 rekur-

3. Borges 1941, 169. 4. Borges 1941, 168. 5. Borges 1941, 168. 6. Borges 1941, 170. 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, 172.

siv, 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.«[9] Und wie die Geschichte dann weiter zeigt, wird er auch beides zugleich gewesen sein – ganz so wie die Version *Original* neben der Version *Original'* existiert.

Es ist kaum nötig zu erwähnen, dass diese Charakteristik des unendlichen Romans, die Borges entwirft, eine ziemlich präzise strukturelle Beschreibung dessen liefert, was eine Software-Versionsverwaltung rund 50 Jahre später bereitstellt: ›ein wachsendes, schwindelerregendes Netz auseinander- und zueinanderstrebender und paralleler Zeiten‹, in dem die verschiedenen Varianten eines Texts nebeneinander, voneinander unabhängig mit jeweils eigenen, offenen Entwicklungshorizonten entstehen. Mit anderen Worten, die vergleichsweise kurze Geschichte der Software und ihren jeweiligen Maßnahmen, Kontrolle über die Quellen zu erlangen oder zu erhalten, korrespondiert – zumindest untergründig – mit der langen Geschichte von literarischer oder gar kollektiver Autorschaft. Man könnte Borges abgründige Geschichte über die »Verzweigung in der Zeit«[10] daher als eine Art literarische Präfiguration der informatischen Versionskontrolle verstehen.

Bevor nun im weiteren Textverlauf von der medialen Praktik des Verzweigens seinerseits wieder verzweigt wird auf die Unterschiedsbestimmung mit *diff*, sei noch ein kleiner Unterzweig eingebunden, der so etwas wie die fundamentale Übertragungsfunktion aller drei Verfahren darstellt. Denn weder *branch* noch *diff* oder *merge* kommen ohne einen Vorgang aus, der die Daten von A nach B schafft; der sie prüft, validiert, transferiert und dupliziert. Es folgt demnach eine kurze Verzweigung zum Vorgang des Kopierens. . .

Abzweigung: Kopieren

Wenn die lange Geschichte des (literarischen) Schreibens über weite Strecken und Genealogien keineswegs das Geschäft einer Einzelperson ist, sondern vielmehr Teamwork, wenn das Schreiben also in den überwiegenden Fällen eine kollektive Tätigkeit darstellt, dann kommt dem Abschreiben oder Kopieren dabei eine besondere Funktionsstelle zu. Zum einen, weil das rasche Vervielfältigen von Texten – vor dem Zeitalter der technischen Reproduzierbarkeit durch Photographie oder Photokopie – nicht selten im Modus des Diktierens stattfindet, hier also zwei interagierende Personen schriftliche Artefakte über das Medium der Stimme wieder in schriftliche

9. Borges 1941, 166/170. 10. Borges 1941, 169.

Artefakte transformieren. Zum anderen, weil selbst beim Abschreiben beispielsweise einer Bibelpassage durch einen einzelnen Mönch im Skriptorium ebenfalls zwei Personen interagieren, insofern die schriftlich fixierte Rede eines Abwesenden das Original darstellt, das durch das Medium des anwesenden Schreibers in Kopie dupliziert wird.

Nun mag man einwenden, dass für die massenhafte Vervielfältigung von Texten seit dem 15. Jahrhundert mit dem Buchdruck ein Medium bereitsteht, dass diese komplizierten Vergegenwärtigungen von Text über Stimme und Handschrift eigentlich überflüssig macht. Das trifft zweifellos zu auf Kopiervorgänge, bei denen ausreichend Zeit zur Anfertigung der Kopien bereit steht. In Situationen allerdings, wo es auf einzelne Minuten ankommt, wo Zeit kritisch, weil knapp, wird oder der Aufwand zur Einrichtung einer Druckvorlage ohnehin viel zu gross wäre, weil es nur einer einzigen Abschrift bedarf, findet die bewährte Form der Vervielfältigung durch Diktat oder eine individuelle Abschrift ihren Einsatz.

Das folgende Szenario rekonstruiert die Funktionen einer hochprofessionalisierten, immer schon international arbeitenden Institution, in der das Vervielfältigen in Serie, der Akt des Kopierens, des Filterns und ggf. noch des Verschlüsselns 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 worden ist.[11] 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, um sie endlich mit einem gefälschten Siegel zu versehen und anschliessend 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. Am Ende des 18. Jahrhunderts residiert diese Reichszentrale Intelligenz-Agentur in unmittelbarer Nähe zur Macht, vis-à-vis zur Wiener Hofburg, um ihrer speziellen 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 so-

11. Leeuw 1999.

> gleich 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.[12]

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 dieser Vorgang 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.«[13] 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. – Diese Operationskette verdient deshalb noch eigens Hervorhebung, weil ihre 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*:[14]

> 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 diktierten 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.[15]

Der Kabinettsdirektor überprüft anschliessend diese derart im beschleunigten Multitasking oder Parallalprocessing gewonnenen Ergebnisse, filtert sie nach der jeweiligen politischen Interessenslage und reicht sie weiter

12. König 1875, 40. 13. Stix 1937, 138. 14. Vgl. dazu Krautter 1982. 15. Stix 1937, 139.

direkt zum Kaiser bzw. zur Polizei.[16] Einen halben Briefbogen pro Minute kennzeichnet eine Datendurchsatzrate, die nicht so viel geringer bleibt als die einer *floppy disk* 200 Jahre später. 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 glichen weniger Postämtern, als Laboratorien.«[17]

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.[18] 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.«[19]

Man muss nicht zwangsläufig verrückt werden oder in schändlicher Absicht arbeiten, wenn es gilt, sich mit einer spezifischen Form der *high fidelity* eines Anderen anzuverwandeln. Eine gesündere und ehrenvollere Form mimetischer Angleichung hinsichtlich der Briefkopien lässt sich zur selben Zeit in der Korrespondenzpraxis von Goethe beobachten, dessen Briefproduktion über die Jahrzehnte rund 20’000 Exemplare umfasst – trotz der expliziten Verweigerung des Geheimrats, die Feder selbst zu führen.

Wie hinlänglich bekannt ist das Aufschreibesystem 1800 keineswegs nur auf die Ausbildung neuer Dichter abgestellt,[20] sondern bedient sich – auch und gerade bei Goethe – eines umfangreichen Apparates von Bedienten, also Sekretären, Schreibern, Kopisten, Kammerdienern und anderen Subalternen aller Art. Dabei ist besonders auffällig, dass alle Subalternen in spezifischer Weise eine Eigenart annehmen, die sie ihrem Herrn und Gebieter ähnlicher werden läßt. Johann Georg Paul Götze, der rund 17 Jahre bei Goethen dient, gelingt es etwa, sich die Handschrift seines Meisters zu

16. Stix 1937, 140. 17. König 1875, 40. 18. König 1875, 34. 19. König 1875, 38. 20. Kittler 1985.

solcher Perfektion anzueignen, dass selbst die Experten später bisweilen Mühe haben werden, sie vom Original treffsicher zu unterscheiden. Zudem übt Götze sich noch darin, zu zeichnen wie sein Vorbild.[21] Die Diener Geist und Stadelmann laufen dagegen mit einem anderen Wahrnehmungsfilter ihres Herrn durch die Welt, oder genauer: in das Theater und durch Steinbrüche. »Alle Diener Goethes haben, jeder nach seinen Möglichkeiten, Züge des äußeren Gehabens ihres Herrn angenommen, sich seine Handschrift angewöhnt und aus seinen Wissensgebieten ihre Steckenpferde gewählt: [Philipp] Seidel philosophische, sprachliche und wirtschaftliche Themen, Geist Botanik, Stadelmann Geologie und Mineralogie, [Michael] Färber Osteologie.«[22] Das hohe Maß an mimetischem Verlangen, der ungestillte Wunsch nach Anverwandlung, zeigt sich jedoch am prägnantesten bei Philipp Seidel, Goethes erstem Subalternen, der später dank seiner Verdienste zum Weimarer Kammerkalkulator befördert wird: »Er hatte sich ihm derart angeähnelt, daß sie ihn Goethes ›vidimirte Kopie‹ nannten«.[23] Seidel versteht sich darauf, den Rededuktus seines Herrn, die Intonation ebenso beiläufig nachzuahmen wie gleich seinem Vorbild den Kopf zu schütteln und sogar dessen »Perpendikulargang« so täuschend echt zu imitieren, »daß man oft versucht war, ihn von weitem für Goethe selbst zu halten.«[24] Der Meister dupliziert sich in seinen Domestiken.

Es wäre zudem irrig anzunehmen, dass Goethe seine Briefe selbst schreibt. Abgesehen vom in grosser Kanzleischrift geschwungenen G. als Signatur setzt er den schon im Original als Kopie durch die nachahmende Hand der Schreiber verfassten Briefe nichts weiter hinzu als gelegentlich Grüsse oder Addenda.[25] Seidel dient zudem auch als historisches Vorbild für den Briefschreieber Richard in Goethes *Egmont*, der die Briefe seines Herrn in einem Akt exzessiver Mimesis auch inhaltlich für seinen Meister ausfertigt.[26] Erst wenn diese eng verflochtene, kollaborative Autorschaft einmal gestört ist, sieht sich G. noch genötigt, selbst zur Feder zu greifen, wenn auch nicht ohne Widerwillen. So klagt Goethe, als 1813 sein zusätzlich zum schreibkundigen Domestiken eingestellter Sekretär Ernst Carl Christian John einmal unpäßlich ist: »Seit vierzehn Tagen hat sich leider meine adoptive rechte Hand kranckheitshalber in’s Bette gelegt und meine angebohrne Rechte ist so faul als ungeschickt, dergestalt daß sie immer Entschuldigung zu finden weis wenn ihr ein Briefblatt vorgelegt wird.«[27]

Über Goethes Praxis des Briefeschreibens und die rekursiven Bezugnahmen, wechselnden Autorschaften und unterschiedlichen Befehlsebenen

21. Schleif 1965, S. 100; Mit dem Bestreben, die Handschrift des Herrn nachzuahmen, stehen Goethes Domestiken keineswegs allein. Auch in den Privatlabors im viktorianischen England, wo die Domestiken zu Laborassistenten werden, findet sich diese Tendenz, so etwa bei Sir William Crookes Diener: »Even Giminghams handwriting became more like Crooke’s.« Gay 1996, S. 330. 22. Schleif 1965, S. 222. 23. Schleif 1965, S. 28. 24. Lyncker 1912, S. 47. 25. Schleif 1965, 40. 26. Vgl. Krajewski 2010, 253–256. 27. Goethe 1887, Nachträge: Briefe, Bd. 51, S. 342.

im Zusammenspiel mit seinen Dienern gäbe es noch viel zu sagen.[28] An dieser Stelle sei der Unterzweig zum grundlegenden *copy*-Befehl jedoch schon wieder verlassen zugunsten der zweiten heuristischen Praktik der Versionskontrolle, dem Abweichen. Denn keine Kopie vor dem digitalen Zeitalter ist authentisch im Sinne allerhöchster Originaltreue oder gar 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 entscheidend, bei der Versionskontrolle über eine Fehlerkorrektur bzw. Validierungsinstanz zu verfügen, die im Zweifelsfall die feinsten Unterschiede zwischen Original und Kopie zu finden verspricht. Diesen kleinen Unterschied macht der *diff*-Befehl (deutlich).

2 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.[29] 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.

Im Vergleich dazu ist der *diff*-Befehl nachgerade ›dumm‹ zu nennen, weil er sich weniger zum Aufspüren stilistischer Differenzen eignet als zur tumben Suche nach Fehlern oder anderen Veränderungen. Denn der Algorithmus geht denkbar einfach vor, indem er ein Mapping, ein Übereinanderlegen von zwei Texten vollführt, um dabei die Lücken und jeweils einseitigen Ergänzungen ausfindig zu machen (Abb. 4).

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,

28. Vgl. Krajewski 2010; sowie Schöne 2015. 29. Der Luhmann-Schüler Peter Fuchs berichtet in seinem Nachruf auf den Meister von einer solchen Form maximaler Distanznahme: »Ich erinnere mich, daß er – ich war noch Student – mir antrug, mit ihm zusammen ein Buch über Reden und Schweigen zu schreiben. Feuer und Flamme, der ich war, schrieb ich ein mächtiges Exposé, meinte mein Bestes, mein Überzeugendstes zu geben. Er schickte mir eine kurze Notiz: ›Anders als Herr Fuchs optiere ich dafür, einfacher anzufangen. . . ‹ Und skizzierte das Projekt unvergleichlich einfacher und punktgenauer auf einem Zettel [. . .] Für mich jedenfalls war dieses ›Anders als Herr Fuchs. . . ‹ der härteste Denkzettel, den ich erhalten habe. Das war nicht ein ›Anders als Sie. . . ‹ oder ›Im Gegensatz zu Ihnen. . . ‹. Das war Extremdistanzierung.« Fuchs 1998, 13.

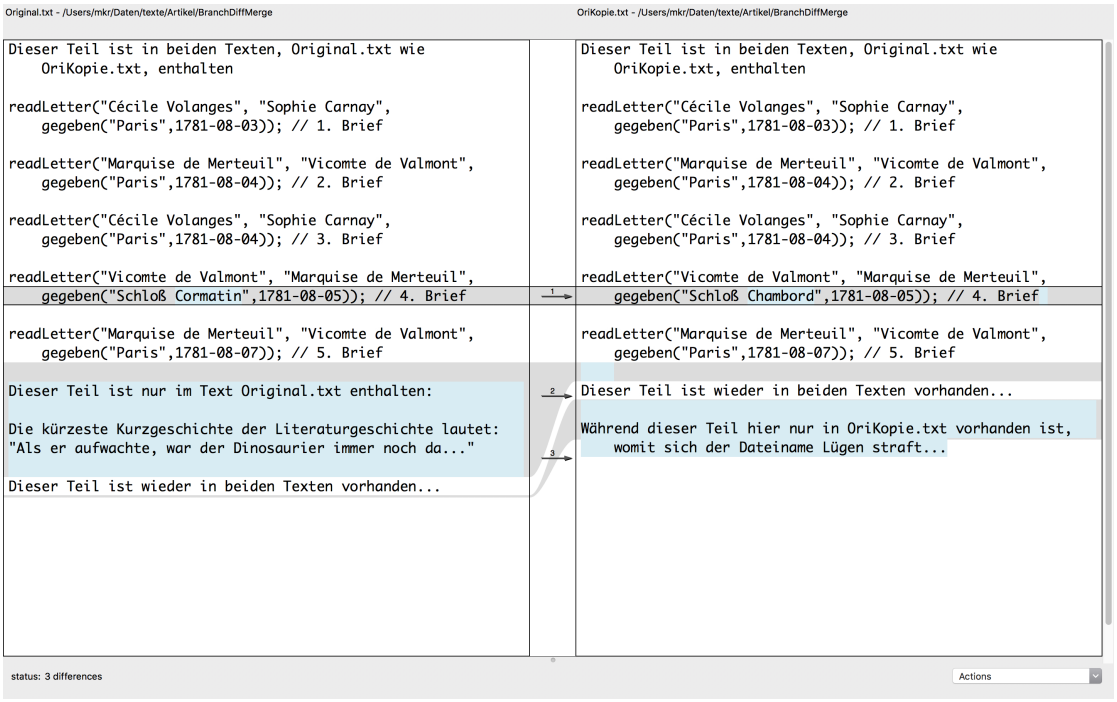

Abbildung 4: *diff* von Original und Kopie vor dem *merge*

wenn nicht infinitesimale Details in der Wort(dar-)stellung? Kurzum, was ist das klassische Pendant zum *diff* und seiner Fixierung der Deltas? »Ich habe Hunderte von Handschriften miteinander verglichen, habe die Fehler korrigiert, die sich durch die Nachlässigkeit der Abschreiber eingeschlichen haben; ich habe den Plan dieses Chaos erschossen, habe die ursprüngliche Ordnung wieder hergestellt«.[30]

Zur zentralen Aufgabe eines Herausgebers oder Bearbeiters einer Edition zählt die Notwendigkeit, den Überblick über die einzelnen Varianten und Entwürfe einer Textfassung zu erhalten, sie auf Stimmigkeit und Korrektheit hin zu überprüfen. Dabei gilt es, jede Version akribisch mit den anderen abzugleichen, eine Kontrolle aller Details in den jeweiligen Varianten vorzunehmen, um auf diese Weise neben allen alternativen Entwürfen eine Sicherung des Textes zu gewährleisten, generiert durch eine Reihe von Entscheidungen, dieser und nicht der anderen Variante den Vorzug zu geben, wobei die philologische Textfassung 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.

30. Borges 1941, 172.

3 Verschmelzen

Die vorgegebene Zeichenanzahl reicht nicht mehr aus, an dieser Stelle noch die dritte Praktik vorzuführen, die grundlegend für die kollaborativen Schreibverfahren sind, die in der Softwareentwicklung und ihrer Versionsverwaltung zur Anwendung gelangen. In der dritten Version dieses Texts ist dieser Abschnitt entsprechend ausführlicher dargestellt. Und zwar ist dort nachzulesen, 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 wird diese Praktik – anders als bei Borges und seinem unendlichen Buch als Labyrinth – zwar ebenso mit einer »prinzipiellen Unendlichkeit«[31], 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. 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 eine algorithmische Struktur annimmt. Vorführen lässt sich an diesem historischen Szenario also ein vordigitaler Merge-Algorithmus, 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 wird daher gezeigt, wie aus dieser informationellen Vereinzelung und Fragmentarisierung durch eine neue Beweglichkeit oder Atomisierung von Informationsbausteinen 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.

Coda: Quellcodekritik

Was mit der Beschreibung medialer Praktiken wie ›Verzweigen‹, ›Kopieren‹, ›Abweichen‹ und ›Verschmelzen‹ bei kollektiven Schreibprozessen – sei es von Software, sei es von Literatur, oder sei es von speziellen Texten wie Katalogen – historisch herzuleiten war, lässt sich ebenso in einen grösseren, methodischen Zusammenhang einordnen. Die Praktiken der Versionskontrolle dienen dabei als eines von zwei Grundelementen inner-

31. Hagen 2004, ??

halb eines wissenschaftlichen Verfahrens, das unter dem heuristischen Begriff ›Quellcodekritik‹ zu fassen wäre. Der Begriff ist ein Kofferwort aus ›Quellcode‹ und der in der Historiographie bewährten, hilfswissenschaftlichen ›Quellenkritik‹.[32] Konzeptionell beinhaltet er einerseits den Bereich der Quellcodeerschliessung, andererseits eine kritische Lektüre der Codes, die in ihrer Dynamik und Veränderbarkeit gleichwohl wie eine historische Quelle aufzufassen sind, welche einer weiteren Einordnung und Kommentierung bedarf.

Unter Quellcodekritik ist also weniger eine neue Forschungsrichtung zu verstehen, wie sie vor rund 10 Jahren die *Critical Code Studies* als programmatische Abspaltung von den *Software Studies* vorgeschlagen haben, um sich darum zu bemühen, Code als Text mit hermeneutischen Verfahren zu lesen.[33] Auch geht es weniger darum, einen Seitenzweig in die *critique génétique* einzuziehen, um die editionsphilologischen Verfahren auf Software zu erweitern.[34] Vielmehr geht es darum, auf einer pragmatischen Ebene eine Methodik (weiter) zu entwickeln, die es erlaubt, Softwarecode nicht einfach nur zur Anwendung zu bringen, sondern ihn systematisch oder punktuell einer kritischen Lektüre zu unterziehen, die Algorithmen also ihrerseits lesbar zu machen durch Kommentare, Reflexionen, Hinweise und gegebenenfalls auch Modifikationen. Dabei zielt diese Methodik nicht allein auf Didaktik, insofern das Verstehen von Programmstrukturen als Stärkung einer digitalgelehrten Kernkompetenz zu begreifen wäre. Das übergeordnete Ziel besteht eher darin, – wie bei der Vermittlung elaborierter Lesefähigkeit mit klassischen akademischen Methoden wie Diskursanalyse, Dekonstruktion oder auch Hermeneutik – durch Training eine Fähigkeit zu erlangen, die der vielbeschworenen Macht der Algorithmen eine Kritik entgegenhält, indem es die einzelnen Programmschritte nachzuvollziehen, offenzulegen, einzuordnen, kontextualisieren und zu erklären vermag. Im Unterschied zu den *Critical Code Studies* geht es dabei nicht darum, Programmstrukturen und Befehlszeilen auf ihre allfälligen Metaphoriken, auf figurative Rede oder ambivalente Bedeutungen hin zu überprüfen, sondern es geht um das extensive Zusammenspiel von Code und Kommentar, um die Algorithmen selbst transparenter und damit nachvollziehbarer zu machen, sie also derart durch erläuternde Kommentare aufzubereiten, dass Interventionen und Modifikationen in den Programmablauf erleichtert werden (vgl. den Kommentar im Hauptfenster von Abb. 5).

Als zentrales Medium der Kritik dient dabei die sogenannte Integrated Development Environment (IDE), also eine Programmierumgebung, die alle wichtigen Hilfsmittel zur Analyse, Recherche, Fehlerrückverfolgung, Code-Produktion und -Verteilung in sich vereint (Abb. 5). Eine IDE ist aber

32. Vgl. dazu z.B. Saxer 2014, 376 ff. Oder Brasi 2004. 33. Vgl. Marino 2006, 2010.
34. Zu letzterem siehe erste Ansätze bei Hiller 2014; zur Editionsphilologie und *critique génétique* allgemein Grésillon 1999.

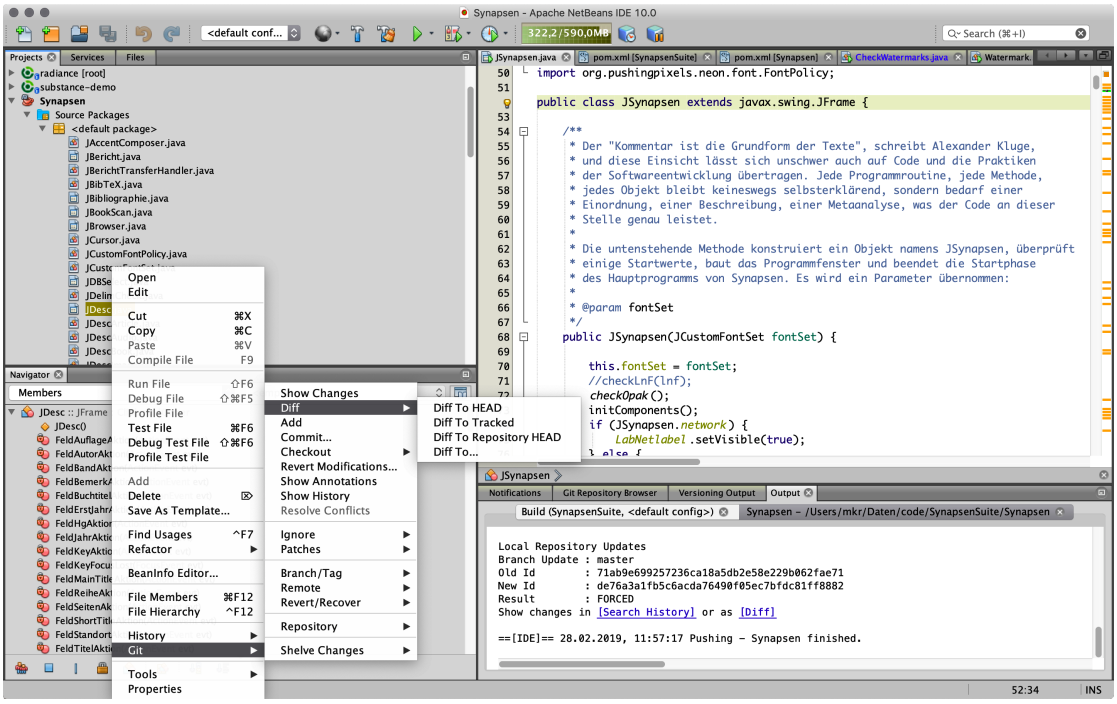

Abbildung 5: Eine *Integrated Development Environment* (IDE) vereint Code, Kommentar, Versionskontrolle und viele andere Funktionen, die zur kollaborativen Softwareentwicklung hilfreich sind

nicht nur die systematische Stelle, wo Fehler korrigiert werden. Es ist zudem der Ort eines aufklärerischen, lektorierenden Korrektorats des Codes (nicht notwendigerweise von einer anderen Person), insofern hier die Kommentare eingefügt werden, die dann zur Dokumentation des Codes weiterverarbeitet werden.[35] Es ist nicht nur der Ort, an dem der Code auf seine Stimmigkeit, Effektivität oder Eleganz hin überprüft und verbessert wird. Darüberhinaus ist es die zentrale Stelle, von wo aus die eigenen Modifikationen an der Software mithilfe der Befehle aus der Versionskontrolle weiterverteilt, oder aber die von anderen modifizierten Teile des Programm in die eigenen lokalen Dateien eingefügt werden. Es ist eine dezentrale Verteilungsplatform, an der die Position des Nutzers beständig oszilliert zwischen Leser und Schreiber. Während man im einen Moment noch den Code in seinen feinsten Verästelungen nachvollzieht, um ihn – wie bei einer hermeneutischen Operation im jeweiligen Teil auf das Ganze zu beziehen – so wird man, um die soeben gewonnene Einsicht nicht zu verlieren, im nächsten Moment zum annotierenden und kommentierenden Leser und damit zugleich zum intervenierenden Schreiber, getreu der

35. Dieses Prinzip, wie es etwa mit Javadoc umgesetzt worden ist, geht auf das Paradigma des *Literate Programming* von Knuth 1984, zurück.

alten rechtshistorischen Maxime, dass der Kommentar die Grundform des Texts und des Codes gleichermassen ist.[36] Eine IDE dient als mediales Milieu von Code, das eine Umgebung bereitstellt, in der die Algorithmen gedeihen, gezüchtet, gehegt, optimiert und schliesslich im Rahmen einer Quellcodekritik nachvollziehbar, verständlich, transparent gemacht werden.

Und schliesslich besteht ein Erkenntnisziel der Quellcodekritik darin, jenseits des Inhaltlichen und der Funktionalität einer Software die Materialitäten der Kommunikation auch im Virtuellen, in diesem Fall also die Mechanismen einer global verteilenden, kollektiven Softwareentwickelung – ganz im Sinne einer klassischen Quellenkritik – auf ihre medialen Praktiken und archivalischen Strukturen hin zu befragen und um die Perspektive ihrer historischen Genese zu erweitern. Eine historische Rekonstruktion von ›branch‹, ›diff‹, ›merge‹ mag dafür lediglich einen Anfang bieten. 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 etwa, dass man beim Programmieren immerzu auf *Bibliotheken* zugreift, auch wenn es sich dabei um *.jar, *.zip, *.dylib, *.lib- oder sonstige Programmbibliotheken handelt, harrt immer noch einer eingehenden kulturwissenschaftlichen Analyse.[37] Dass die weitestgehend geschichtsvergessene *computer science* auf die Historizität von Algorithmen, Programmiersprachen,[38] den darin verwendeten Metaphern und Bezeichnungen, aber auch den Entwicklungsumgebungen selbst – jenseits von Kompatibilitätsfragen – für gewöhnlich wenig Aufmerksamkeit richtet, mag wohl unabänderlich sein. Dass aber die kulturwissenschaftlich angeleiteten *software studies* hier noch lohnende Untersuchungsgebiete finden können, liegt auf der Hand.

Die Befehle *branch*, *copy*, *diff*, *merge* zählen zu den eminenten Funktionen verteilter Code-Autorschaft in der Softwareentwicklung. Wie zu zeigen versucht worden ist basieren diese Funktionen auf Praktiken, die sich zum einen in der Literaturgeschichte und ihrem Zusammenspiel aus experimenteller Autorschaft und Editionstätigkeit finden (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 kollaborativen 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

36. Vgl. Krajewski und Vismann 2009. 37. Einen eher künstlerischen Anatz, die Leistungen einer Programmbibliothek zu reflektieren, findet sich bei Harwood 2008, weitergehende, die Metaphorik von ›Bibliothek‹ und dem gesamten ›Bibiliothekswesen‹ kritisch analysierende Untersuchungen, die Reichweite, Passgenauigkeit und Erkenntnisgewinn eines solchen semantischen Feldes im Bereich der Softwareentwicklung bilanzierten, sind im Moment noch ein Desiderat. 38. Vgl. Hagen 1997.

informatische Versionsverwaltung verfertigt ist – ohne es zu wissen. Es bleibt einer Methode wie der Quellcodekritik überlassen, die aktuelle Funktionsweise der *computer science* historisch zurück zu verfolgen, um *in der Tiefe der Geschichte* nach Denkfiguren und Funktionsweisen, nach medialen Praktiken und historischen Arbeitsweisen, nach Fragestellungen und Problemlösungen zu suchen, mit denen die Funktionen und Praktiken der gegenwärtigen Softwareentwicklung erläutert, ergänzt und nicht zuletzt erweitert werden können.[39]

Literaturverzeichnis

Borges, Jorge Luis. 1941. »Der Garten der Pfade, die sich verzweigen«. In *Der Erzählungen erster Teil,* 1:161–173. Gesammelte Werke. München: Carl Hanser Verlag, 2000.

Brasi, Lucas. 2004. *Die erfundene Antike. Einführung in die Quellenkritik.* 2. Auflage. Hamburg: UBW.

Fuchs, Peter. 1998. »Man muß schmunzeln können«. *die tageszeitung* (14. November): 13–14.

Gay, Hannah. 1996. »Invisible resource: William Crookes and his circle of support, 1871–81«. *The British Journal for the History of Science* 29:311–336.

Goethe, Johann Wolfgang. 1887. *Goethes Werke.* Weimarer Ausgabe. Weimar: Böhlau Verlag, 1919.

Grésillon, Almuth. 1999. *Literarische Handschriften. Einführung in die critique génétique.* Bern: Peter Lang Verlag.

Hagen, Wolfgang. 1997. »Der Stil der Sourcen«. In *Hyperkult,* herausgegeben von Georg Christoph Tholen Wolfgang Coy und Martin Warnke, 33–68. Basel u.a.: Stroemfeld.

———, Hrsg. 2004. *Warum haben Sie keinen Fernseher, Herr Luhmann? Letzte Gespräche mit Niklas Luhmann.* Berlin: Kulturverlag Kadmos.

Harwood, Graham. 2008. »Class Library«. In *Software studies. A lexicon,* herausgegeben von Matthew Fuller, 37–39. Cambridge, Massachusettes und London: MIT Press.

Hiller, Moritz. 2014. »Diskurs/Signal (II). Prolegomena zu einer Philologie digitaler Quelltexte«. *editio. Internationales Jahrbuch für Editionswissenschaft* 28:192–212.

39. Wie das konkret aussehen könnte, wird in der dritten Fassung dieses Texts detaillierter entwickelt.

Kittler, Friedrich. 1985. *Aufschreibesysteme 1800 · 1900.* 3., vollständig überarbeitete Auflage. München: Wilhelm Fink Verlag, 1995.

Knuth, Donald E. 1984. »Literate Programming«. *The Computer Journal* 27:97–111.

König, Bruno Emil. 1875. *Schwarze Kabinette. Mit Anlagen: Geschichte der Thurn und Taxis'schen Postanstalt und des österreichischen Postwesens, und ueber die gerichtliche Beschlagnahme von Postsendungen in Preussen-Deutschland.* Braunschweig: Bracke.

Krajewski, Markus. 2010. *Der Diener. Mediengeschichte einer Figur zwischen König und Klient.* S. Fischer Wissenschaft. Frankfurt am Main: S. Fischer Verlag.

Krajewski, Markus, und Cornelia Vismann. 2009. »Kommentar, Code und Kodifikation«. *Zeitschrift für Ideengeschichte* Frühjahr 2009:5–16. Themenheft »Kommentar«.

Krautter, Konrad. 1982. »Acsi ore ad os ... Eine mittelalterliche Theorie des Briefes und ihr antiker Hintergrund«. *Antike und Abendland* 28 (2): 155–168.

Leeuw, Karl de. 1999. »The Black Chamber in the Dutch Republic During the War of the Spanish Succession and its Aftermath, 1707-1715«. *The Historical Journal* 42 (1): 133–156.

Lyncker, Karl Wilhelm Heinrich von. 1912. *Am Weimarischen Hofe unter Amalien und Karl August. Erinnerungen.* Mittlers Goethe-Bücherei. Berlin: Ernst Siegfried Mittler und Sohn.

Marino, Mark C. 2006. *Critical Code Studies.* electronicbookreview.com. http://electronicbookreview.com/essay/critical-code-studies/.

———. 2010. *Critical Code Studies and the electronic book review: An Introduction.* http://electronicbookreview.com. http://electronicbookreview.com/essay/critical-code-studies-and-the-electronic-book-review-an-introduction/.

Saxer, Daniela. 2014. *Die Schärfung des Quellenblicks. Forschungspraktiken in der Geschichtswissenschaft 1840–1914.* München: De Gruyter Oldenbourg.

Schleif, Walter. 1965. *Goethes Diener.* Bd. 17. Beiträge zur Deutschen Klassik. Berlin, Weimar: Aufbau-Verlag.

Schöne, Albrecht. 2015. *Der Briefschreiber Goethe.* München: C.H. Beck.

Stix, Franz. 1937. »Zur Geschichte und Organisation der Wiener Geheimen Ziffernkanzlei«. *Mitteilungen des Osterreichischen Instituts fir Geschichtsforschung* 51:131–160.

Yuill, Simon. 2008. »Concurrent Versions System«. In *Software studies. A lexicon,* herausgegeben von Matthew Fuller, 64–69. Cambridge, Massachusettes und London: MIT Press.