Funktionale und deklarative Programmierung-basierte Methode für nachhaltige, reproduzierbare und verifizierbare Datenkuration.

Barabucci, Gioele
gioele.barabucci@uni-koeln.de
Universität zu Köln, Deutschland

Einleitung

Durch die wachsende Menge an digitalen Daten im Bereich Digital Humanities bedarf es zunehmend der Arbeit von Datenkuratoren. Aufgrund der ständigen Steigerung der Zahl und des Umfangs der zu verarbeitenden Quellen ist jedoch der übliche Modus Operandi der Datenkuratoren (manuelle Konversionen und konsekutive Anpassungen) untragbar geworden.

Dieser Vortrag stellt eine neue Methode für die Kuration digitaler Daten vor, die auf den Prinzipien der funktionalen Programmierung, der unix-Tools und der XML-Technologien basiert. Diese Methode wurde vom Cologne Center for eHumanities der Universität zu Köln seit 2014 im Rahmen des Lazarus-Projekts (CCeH 2014) und danach in verschiedenen anderen DH-Projekten angewendet.

Die Besonderheit dieser Methode besteht in der Aufgliederung des Kurationsprozesses in eine Pipeline von Miniprogrammen, von denen jedes einzelne einen präzisen Schritt des Kurationsprozesses darstellt. Wird ein Fehler in den resultierenden Dateien bemerkt, kann ein neuer Schritt geschrieben werden und in die Kette eingebaut werden, oder die existierenden Schritte korrigiert werden. Anschließend wird die Kurationspipeline neu laufen gelassen.

Der Hauptgrundsatz lautet: Keine Datei wird „manuell“ modifiziert, jede Operation muss von einem Miniprogramm ausgeführt werden.

Konsequenz dieses Grundsatzes und Hauptvorteil der vorgestellten Methode ist, dass die ganze Arbeit des Kurators — mitsamt seiner Entscheidungen und seiner bevorzugten Arbeitsweisen — in dieser Pipeline von Miniprogrammen dokumentiert ist und in sie eingebettet ist. Des weiteren sind der Kurationsprozess und seine Ergebnisse einfach zu reproduzieren und zu verifizieren. Das führt zu einer besseren Nachvollziehbarkeit der Kurationsarbeit, auch wenn die Kuration von einem Team durchgeführt wird.

Kuration von Digitalen Daten

Eine der Aufgaben der Datenkuration und der Datenkuratoren ist: »[to] intervene in the research process in order to translate or migrate data into new formats, to enhance it through additional layers of context or markup, to create connections between data sets, and to otherwise ensure that data is maintained in as highly-functional a form as possible.« (Flanders & Muñoz 2017).

Praktisch können wir die Arbeit der Datenkuratoren auf folgende Art und Weise grob zusammenfassen:

- Die Daten werden von den Forschern zu den Kuratoren übertragen.
- Die Kuratoren studieren die Daten, sowohl ihren Inhalt als auch ihr Format.
- Die Kuratoren reichern die Daten mit den nötigen Metadaten an und sie sorgen dafür, dass eventuelle Inkohärenzen zwischen den originalen Formaten und den Zielformaten ausgeglichen werden.
- Die so verarbeiteten Daten werden archiviert, publiziert oder in anderen Projekten verwendet.

Datenkuration am CCeH: Das Cologne-Sanskrit-Lexicon-Projekt

Ein praktisches Beispiel von Datenkuration sind die Wörterbücher des Cologne Sanskrit Lexikons. Diese wurden von verschiedenen Wissenschaftlern der Universität zu Köln seit den 90er Jahren (pre XML und pre Unicode) erarbeitet, innerhalb des Lazarus-Projektes vom CCeH kuratiert (d.h. in TEI/Unicode umgewandelt) und 2015 zugänglich gemacht.

Die anfängliche Arbeitshypothese, welche im Verlauf jedoch fallen gelassen wurde, sah einen eher klassischen Workflow vor: Die originalen Dateien in XML umwandeln, danach eine XSLT-Transformation nutzen, um diese in TEI umzuwandeln und schließlich die durch die Transformation entstandenen kleinen Fehler per Hand verbessern.

Wir haben es vorgezogen, diesen Weg aus zwei Gründen nicht einzuschlagen.

Erstens: Von Beginn an sind verschiedene Versionen der zu kuratierenden Dateien aufgetaucht. Hätten wir in der Zwischenzeit neue Versionen der Dateien entdeckt, wäre die bis dahin geleistete Arbeit vergebens gewesen.

Zweitens: Die Arbeitsgruppe bestand aus drei Personen aus unterschiedlichen Fächern und mit unterschiedlichen Herangehensweisen an das Thema Kuration. Einen einheitlichen Stil bei zu behalten wäre nicht einfach — wahrscheinlich unmöglich — gewesen.

Die Arbeitsgruppe hat sich dann für eine andere Arbeitsweise entschieden: Eine „wiederholbare Pipeline“-Methode, die auf der funktionalen Programmierung basiert, statt manueller Konversionen und Anpassungen.

In dieser Methode ist jeder Arbeitsschritt formell durch ein sehr kleines Programm beschrieben, das in

einer funktionalen und deklarativen Programmiersprache implementiert ist und im Durchschnitt aus weniger als 15 Instruktionen besteht. In dieser Art sind sehr unterschiedliche Schritte implementiert, z.B. die Normalisierung der Lemmata, die Behebung von technischen Fehlern, die Integration von externen Quellen.

Diese Programme, die die Schritte der Kurationsarbeit darstellen, sind im Sinne einer Pipeline organisiert, d.h. der Output des einen ist der Input eines anderen. Die Kuratoren erklären, welches die kommenden Schritte sind und welche Abhängigkeiten zwischen den einzelnen Schritten bestehen (z.B. dass die Umwandlungsschritte dem Abrufschritt folgen sollen). Der folgende Abschnitt enthält verschiedene konkrete Beispiele von Kurationspipelines und Schritten/ Miniprogrammen.

Alle diese Schritte sind reine *idempotente* Funktionen, d.h. dass ihr Ergebnis nur von den Input-Daten anhängig ist. Konkret bedeutet das, dass man die Kurationspipelinemehrmals durchlaufen kann, und immer das gleiche Ergebnis erhält. Dies steht im Gegensatz zu den klassischen Skript-basierten Methoden.

Lazarus-Kurationsworkflow: XML-Pipelines, Makefiles und Schematron

In der Essenz bedeutet das konkret, dass der Kurationworkflow, der im Lazaruz-Projekt und in anderen folgenden CCeH-Projekten benutzt wurde, aus drei großen Komponenten besteht:

1. Die Makefiles. Ein Makefile ist eine Datei, die den gesamten Kurationsprozess mittels des unix-Tools make steuert. Der Makefile erklärt, wie man eine Datei X (genannt **Target**) durch die Dateien A, B und C (genannt **Abhängigkeiten von X**) herstellen kann. Im Fall des Cologne-Sanskrit-Lexicon-Projekts erklären die Makefiles wie man die target TEI-Deteien herstellen und testen kann, d.h. wo man die originalen Dateien mit den Sanskrit-Wörterbüchern finden kann, wie man sie herunterladen kann, wie man den Konversionprozess durch die Konversionspipelines durchführen kann usw.
2. Die Konversionspipelines. Jede Pipeline ist für die Konversion bestimmter Dateien verantwortlich und besteht aus verschieden Schritten. Jeder Schritt ist implementiert durch eine XSLT-Transformation (die funktionalen Miniprogramme, wie oben erwähnt). Die Pipelines selbst sind XProc-basierte XML-Pipelines (Walsh 2007).
3. DieTests. Verschiedeneautomatisierte Schematron-basierte Tests kontrollieren, dass die hergestelltenDateien valide sind, dass keine altenschon behobenen Fehler erneut eingepflegt werden, sowie dass keine Informationen verloren gehen.

Um die Methode und die Beziehungen zwischen den verschieden Komponenten besser zu verstehen, stellen wir hier ein konkretes Beispiel vor: Das Monier Sanskrit-English Wörterbuch, Teil des Cologne-Sanskrit-Lexicon-Projekts.

Die ursprünglichenDateien mit den Digitalisatenund den Transkriptionen des Monier-Wörterbuches, Nachlass der Arbeit von Thomas Malten et al., sind auf einem Server der Universität zu Köln gespeichert und archiviert. Um dieses Wörterbuch in dasneue Cologne Sanskrit-Lexicon zu integrieren, müssen die Kuratoren folgenden Operationen durchführen:

1. Die Originaldateien vom Server abrufen;
2. Kleine Markup-Fehler beheben;
3. Die Daten in TEI/XML umwandeln;
4. Verweise zu externen Datenbanken/Quellen integrieren;
5. Prüfen, ob Fehler unterlaufen sind, bzw. dass kein Lemma verloren wurde;
6. Die kuratiertenDaten zur Verfügung stellen, sodass sie auf den Produktionsserver hochgeladen werden können.

Diese Operationen, die denKurationsprozess grob zusammenfassen, sind imMakefile monier.mkbeschrieben. Die Operationensind im Makefile in der folgenden Form ausgedrückt: „1) Datei X wird ausgehend von Datei Y hergestellt. 2) Wenn X fehlt oder älter als Y ist, wird X hergestellt, indem man Y anProgramm K mit bestimmten Parametern übergibt“. Abbildung 1 zeigt einige Regeln, die im Vergleich zum Original vereinfacht dargestellt sind.

```
STEPS = $(wildcard $(XPROC_ROOT)/*.xsl)
PIPELINE = (XPROC_ROOT)/monier/conversion.xpl
EXTRAS += monier/abbreviations.tei
EXTRAS += monier/authorities.tei
EXTRAS += monier/authorities-links.tei
EXTRAS += monier/greek.tei

monier.tei: $(STEPS)
monier.tei: $(PIPELINE) | $(XPROC_EXECUTABLE)
monier.tei: $(EXTRAS)
monier.tei: monier.xml
        $(XPROC) -isource=$< result-url=$(abspath $@) $(PIPELINE)

monier.split.tei: $(STEPS)
monier.split.tei: $(PIPELINE) | $(XPROC_EXECUTABLE)
monier.split.tei: $(EXTRAS)
monier.split.tei: CHUNK = 1 # Overridable at runtime
monier.split.tei: monier.split-$(CHUNK).xml
        $(XPROC) -isource=$< result-url=$(abspath $@) $(PIPELINE)
```

Abbildung 1: Makefile für die Kuration des Monier Wörterbuches. In den ersten Zeilen werden verschiedene Parameter eingerichtet. Dann werden die Abhängigkeiten der target-Datei monier.tei beschrieben. Schließlich wird das Kommando eingerichtet, das man benötigt, um die target-Datei herzustellen.

Die Makefiles geben in *imperativer*Weise vor, welche Daten die *funktionale*Pipeline durchlaufen sollen und wo das finale Ergebnis gespeichert werden soll. Dies macht unter anderem das Testen der Kurationspipeline anhand eines Auszugesdes Wörterbuches möglich,

ohne die Pipeline an sich zu verändern; es werden lediglich Änderungen einiger Parameter in den Makefiles vorgenommen.

Operationen 2, 3 und 4 (die Behebung von Fehlern, die Umwandlung der originalen Daten in TEI/XML und deren Verlinkung mit externen Datenbanken) sind das Herzstück des Kurationsprozesses und wird durch verschiedeneXProcPipelines durchgeführt. Diese Pipelinesbestehen insgesamt aus 35 verschiedenen XSLT-Transformationen, von denen jede einzelne einen spezifischen Schritt des Kurationsprozesses darstellt. Beispiele für diese Schritte sind:fehlplatziertes Markup verschieben, falsch kodierte Devanagari-Buchstaben richtig stellen, bibliographische Referenzen hinzufügen. Abbildungen2 und 3 zeigen Ausschnitte von zwei XSLT-Transformationen.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">
  <xsl:import href="../common/identity.xsl"/>
  <xsl:import href="../common/chars.xsl"/>

  <!--
    This transformation reworks the placement of `;` and
    whitespace in cases where there is more than one `<ls>`.

    The result is more similar to the facsimile.
  -->

  <!--
    Turns <ls>Ka1t2h. ; </ls><ls>Anukr.</ls>
    into  <ls>Ka1t2h.</ls>; <ls>Anukr.</ls>
  -->

  <xsl:template match="ls[ends-with(.,' ; ')]">
  <xsl:variable name="first-part"
                select="concat('', substring-before(., ' ; '))"/>

  <ls>
    <xsl:value-of select="$first-part"/>
  </ls>
  <xsl:text>;</xsl:text>
  <xsl:value-of select="$char-space"/>
</xsl:template>
```

Abbildung 2: Ein Miniprogramm. Dieser Schritt korrigiert nur einen bestimmten Fehler.

```
<!-- ls elments preceded by abE are ignored:
     they have already been used inside abE -->
<xsl:template match="ls[preceding-sibling::node()[1][self::abE]]"/>

<!-- Literary quotes and sources -->
<xsl:template match="ls">
  <cit type="literary_source">
    <bibl xml:lang="{$csdl-language}">
      <ref>
        <xsl:apply-templates/>
      </ref>
    </bibl>
  </cit>
</xsl:template>
```

Abbildung 3: Auszug aus der Haupt-TEI-Transformation. Da die vorherigen Schritte die kleinen Fehler schon behoben haben, kann diese Transformation kurz und bündig sein.

Diese Miniprogramme lesen oderschreiben keine Datei. Sie fungieren als rein funktionale Filter, die die Daten empfangen, einige Anteile modifizieren, und die modifizierten Daten zurückgeben. Wie bereits erwähnt ist die Aufgabe des Makefiles zu entscheiden, welche Daten in die Pipeline eingespeist werden sollen und wo die Ergebnisse gespeichert werden sollen. Die Pipeline kümmert sich nicht um diese Details. Dies verringert den Aufwand für die Entwicklung der Miniprogramme erheblich.

Die Miniprogramme, aus denen die Pipeline besteht, spiegeln die Entscheidungen des Teams wider, das diese Daten kuratiert hat (Peter Dängeli, Martina Gödel und Gioele Barabucci, mit der wissenschaftlichen Kollaborationvon Felix Rau). Weil die Entwicklung des Codes der Pipeline und der Miniprogramme mittels eines Repositorium auf GitHub stattgefunden hat,ist es möglich, den Entwicklungsprozessder Kurationsarbeit nachzuvollziehen. Insbesondere ist es möglich, zu sehen, wie einzelneSchritte,die sich alsirrtümlich herausgestellt haben, durch bessere Schritte ersetzt werden, ohne das die übrigen Teile der Pipeline verändert werden müssen.

Schließlich Operation 5 (testen, dass kein Fehler untergelaufen ist)ist durch Schematron-basierte Testsimplementiert. Während dieser Operation wird getestet, 1) dass der Umwandlungsprozess keine Daten unabsichtlich entfernthat und, 2) dass alte Fehler, die schon korrigiert worden sind, wiedereingeführt wurden. Die Wiedereinführung von alten Fehlernist ein Ereignis, das in über Jahre andauerndenProjekten leider häufig vorkommt. Diese Art von Test ist von den Best Practices der Continous Integration in der Softwareentwicklung inspiriert.

Vorzüge der vorgestelltenMethode

Der Gebrauch dieser Methode hat viele Vorteile, sowohl im Hinblick auf die methodologische Stringenz als auch die Technik an sich:

- Jede einzelne Handlung der Kuratoren ist formalisiert und dokumentiert (durch XSLT-Code und Code-Kommentare). Die geringe Größe der Miniprogramme macht deren Code praktisch selbst dokumentierend.
- Jeder Wissenschaftler kann unabhängig nachvollziehen, wie die Ergebnisse entstanden sind.
- Jeder Schritt kann einzeln getestet werden.
- Man kann zurückverfolgen, welcher Schritt ein bestimmtes Konstrukt in den Ergebnissen generiert hat.
- Die Wiederverwendung von Schritten ist möglich und leicht nachzuvollziehen.
- Eine methodologische Kohärenz kann über Jahre hinweg beibehalten werden, auch wenn neue Kuratoren diese Daten verwalten werden.
- Dank der Speicherung verschiedener Schritte in Versioning-Systemen wie Git, kann man sehen, wie der Kurationsprozess entwickelt worden ist.

Die Hauptregel dieser Methode, dass keine Datei „manuell“ modifiziert wird und alles durch Miniprogramme ausgeführt wird, garantiert, dass jede Operation an den gegebenen Daten klar definiert und ausdrücklich formuliert ist.

Die Rolle der Kuratoren

Diese Methode ändert nicht die Rolle oder die Verantwortung der Kuratoren, aber sie verändert grundlegend ihre tägliche Arbeit. Die Arbeit der Kuratoren besteht nicht mehr in dem Modifizieren von Dateien in einem Editor, sondern in dem Schreiben von Arbeitsschritten und in der korrekten Verwaltung von den Abhängigkeiten zwischen Arbeitsschritten.

Die Kurationsarbeit ist dann in zwei Teile aufgeteilt. Der erste Teil besteht im Schreiben und in der schrittweisen Präzisierung von Kurationsprogrammen, welches die Hauptaufgabe der Kuratoren darstellt. Hierin zeigt sich die Fähigkeit, die Erfahrung der Kuratoren sowie die von ihnen präferierten anwendbaren Richtlinien. Der zweite Teil ist die Generierung von kuratierten Daten, welche in sterilerArt und Wiese von einemKoordinationsprogramm vollzogen wird. Es führt die verschiedenen Schritte in der von Kuratoren bestimmten Reihenfolge innerhalbwenigerMinuten durch.

Eine letzte wichtige Auswirkung dieser Methode ist, dass was registriert/gespeichert wird, nicht nur die Endergebnisse sind, sondern der ganze Kurationsprozess: Vom Abrufen der originalen Daten bis zu der Speicherung der kuratierten Daten. Es besteht die Möglichkeit diesen Prozess der Öffentlichkeit zugänglich zu machen, nicht nur um eine bessere Transparenz zu schaffen, sondern auch um die Zusammenarbeit mit externen Kuratoren zu erleichtern.

Zusammenfassend ergibt sich das Besondere dieserMethode aus der Tatsache, dass nicht nur das Ergebnis der Kuration, sondern auch der Kurationsprozess dokumentiert wird.

Verwandte Arbeiten

Für die Kuration digitaler Daten wurde oft ein einfachererWorkflow vorgeschlagen: die Daten ändern und danach die Ergebnisse in einem Git-Repositorium speichern. (Reeve 2016, Crowley et at. 2017). Die Idee der Befürworter dieses Workflows ist, dass die Speichrung des Datenstatuses nach jedem Arbeitsschritt den Kurationsprozess adäquat widerspiegeln. Das greift jedoch zu kurz. Git speichert nur was geändert wurde, nicht mit welcher Absicht eine Änderung durchgeführt wurde. Natürlich könnten diese Absichten und die entsprechenden Begründungen in einer Commit-Nachricht beschrieben werden, aber oft sind sie es nicht und in jedem Fall können Commit-Nachrichten nicht so präzise wie ein Stück Code sein. Des weiteren löst die Änderungen mithilfe von Git nachzuvollziehen nicht die Probleme, welche entstehen, wenn die originalen Daten verändert werden: In diesem Fall muss die ganze Arbeit von vorne begonnen werden.

Workflows wie der in diesem Beitrag beschriebene, in welchen die Hauptaufgabe der Kuratoren ist, Pipelines zu schreiben, finden sich häufig in der Informatik (Doltra & Löh 2008; Schoen & Perry 2014) und in der Physik (Peng 2009).

Diese sind auch im Bereich Digital Scholarly Editing vorgeschlagen worden, z.B. von van Zundert (2016) oder Barabucci und Fischer (2017).

Bibliographie

Barabucci, Gioele / Fischer, Franz (2017): „The formalization of textual criticism: bridging the gap between automated collation and edited critical texts“, in: Advances in Digital Scholarly Editing: Papers presented at the DiXiT conferences in The Hague, Cologne, and Antwerp. Sidestone Press.

CCeH (2014). „sanskrit-dict-to-tei: TEI-fy existing Sanskrit dictionaries.“, https://github.com/cceh/sanskrit-dict-to-tei (Das Repository wird im Laufe des Jahres 2018 veröffentlicht werden).

Crowley, Ronan / Reeve, Jonathan / Schäuble, Johannes (2017). open-editions/corpus-joyce-ulysses-tei: Zenodo release (Version v0.1.1). Zenodo. 10.5281/zenodo.583139

Dolstra, Eelco / Löh, Andres (2008). „NixOS: A purely functional Linux distribution“, in: ACM Sigplan Notices, 43(9): 367-378.

Duvall, Paul M. / Matyas, Steve / Glover, Andrew (2007). „Continuous integration: improving software quality and reducing risk“. Pearson Education.

Flanders, Julia / Muñoz, Trevor (2017). „An Introduction to Humanities Data Curation“, http://guide.dhcuration.org/contents/intro/ [letzter Zugriff 2018-01-10].

Peng, Roger D (2009). „Reproducible research and biostatistics“, in: Biostatistics, 10(3): 405-408.

Reeve, Jonathan (2016). „Git-Lit: an Application of Distributed Version Control Technology toward the Creation of 50,000 Digital Scholarly Editions“, in: Digital Humanities 2016: Conference Abstracts. Jagiellonian University & Pedagogical University, Kraków, pp. 657-658.

Schoen, Seth / Perry, Mike (2014). „Why and how of reproducible builds: Distrusting our own infrastructure for safer software releases“, https://air.mozilla.org/why-and-how-of-reproducible-builds-distrusting-our-own-infrastructure-for-safer-software-releases/ [letzter Zugriff 2018-01-10].

Walsh, Norman (2007). „XProc: An XML pipeline language“ in: XML Prague 2007.

van Zundert, Joris J. (2016). „Close Reading and Slow Programming — Computer Code as Digital Scholarly Edition.“, in: ESTS 2016.