Literate Programming und Dynamisches Publizieren

eingereicht von Tino Fritsch, Gruppe II (05.03.2020)

Bezugnehmend auf einige Gedanken im Beitrag „Dynamic Publishing Formats and Collaborative Authoring“ [1], soll im vorliegenden Text Donald E. Knuths „Literate Programming“-Paradigma vorgestellt werden. Daraus resultierende Praktiken für das dynamische Publizieren im wissenschaftlichen Kontext sollen dabei anhand der Anforderungen an Dateiformate (Verfügbarkeit / Beherrschbarkeit, Konvertierbarkeit / Austauschbarkeit, Präsentierbarkeit, Recherchierbarkeit, Strukturierbarkeit, Archivierung / Standardisierung [Präsentation Horstmann W., Elektronisches Publizieren und Open Access, Abschnitt 2.2]) überprüft werden.

Heller et al. diskutieren in ihrem Beitrag die Vorteile digitaler, dynamischer Publikationen gegenüber herkömmlichen Veröffentlichungen anhand der Begriffe *Kollaborative Autorschaft*, *Reuse* und *Remix*. [1, S.194] Kollaborative Texterstellung kann einerseits als das gemeinsame Arbeiten an einem Text innerhalb einer Gruppe sowie als wechselseitige Verwendung fremden Text- und Datenmaterials nach Open-Access- und Open-Science-Standards [2] verstanden werden. Redundanzen in der Textproduktion (bzw. im Forschungsprozess allgemein) sollen so minimiert, die Verwendung einmal erhobener Daten für weitere Forscher und Nutzer ermöglicht werden. Dabei ist anzunehmen, dass die Textproduktion über zeitliche und organisatorische Abstände hinweg erfolgt. Aus Perspektive eines Autors sollte im Kontext dynamischer Publikationen und im Hinblick auf Reuse und Remix sichergestellt werden, dass zum einen, dem Autor organisatorisch (Institut, Forschergruppe, ...) nahestehenden Forscherkollegen Fließtext, Daten und Quellcode zugänglich gemacht werden, zum anderen können anonyme Leser (und User) aus dynamischen Publikationen einen Mehrwert erzielen, etwa indem Fließtext durch Codeblöcke angereichert wird, Versionierungen nachvollziehbar sind und interaktive oder zeitbasierte Elemente eingebunden werden (und diese Elemente genutzt und rekombiniert werden können).

Das von Donald E. Knuth formulierte Paradigma des Literate Programming fokussiert auf die Herstellung lesbaren Quellcodes. Genauer: Fließtext („Prose“) und Programmiersprache (Code) sind mindestens gleichwertig zu behandeln, in dem Sinne, dass im Code ausgeführte logische Strukturen und Algorithmen in einer für Menschen (und Nicht-Informatiker) verständlichen Sprache ausformuliert werden. Programmcode und Fließtext sind dabei notwendig in einer einzigen Datei enthalten, können als diskrete Abschnitte vermischt und rekombiniert werden, was auch bedingt, dass der Quellcode keine sequenzielle Struktur aufweisen darf, sondern von der Programmierumgebung aus den einzelnen Blöcken funktionstüchtig zusammengesetzt wird.

Wirkmächtig sind dabei die Funktionen „tangle“ und „weave“: Erstere verwickelt die einzelnen Programmcodeblöcke zu einem ausführbaren Gesamtprogramm, letztere Funktion verwebt Textblöcke zu einem Gesamttext. [3] Diese knappe Beschreibung der Prinzipien des Literate Programming kann – grob wiedergegeben – nach Childs [4, S.7] anhand dieser Charakteristika bzw. Anforderungen zusammengefasst werden:

- Code und Fließtext sind in einem einzigen Dokument enthalten,
- Code und Fließtext verhalten sich komplementär,
- Unterteilung des Dokuments in „logische“ (im Sinne von: sinnvoll gegliederte) Einheiten,
- Anordnung der Blöcke nach inhaltlichen Überlegungen (und nicht nach Erfordernissen des Codes),
- Verweis auf alternative Möglichkeiten der Umsetzung des Codes und Rücksicht auf zukünftige Wartung und Erweiterbarkeit,
- Umfassende Beschreibung der Problemstellung und -lösung (textlich, mathematisch, grafisch),
- Querverweise, Indizes, Keywords, Variablennamen etc. sollten möglichst sinnfällig und automatisch erzeugt sowie gut dokumentiert sein.

Obwohl Knuth mit der Programmiersprache WEB und dem darauf basierenden TeX durchaus eine technische Infrastruktur bereitgestellt hat, ist Literate Programming letztlich als (Design-)Methode zu verstehen, als „disciplined way of doing what we should be doing“. [4, S.3]

Liest man Knuths Paradigma (Methode?) des Literate Programming nun weniger aus der Perspektive des um "Lesbarkeit" bemühten Informatikers, sondern aus der Warte des programmierenden und datennutzenden (Sozial-)Wissenschaftlers, erschließt sich ein ideelles Grundgerüst um Datenmaterial, Aufzeichnungen und Visualisierungen in eine dynamische Publikation einzubinden. Ein konstruiertes Beispiel: Ein sozialwissenschaftlicher Text könnte eine Datenbankabfrage beinhalten, welche aktuelles Datenmaterial einer Open-Data-Initiative abruft, einige Abschnitte weiter unten im Text mittels der Statistiksoftware/-sprache R die gezogenen Daten auswertet und weiter unten im Dokument wird darauf basierend ein dynamisch erstelltes Diagramm präsentiert. Dabei gehen Möglichkeiten und Notwendigkeiten parallel: die Verwendung von Code verlangt dessen Dokumentation, Zugänglichkeit und damit Nachvollziehbarkeit bzw. Qualitätskontrolle (gleiches gilt für das jeweilige Datenmaterial). Letztlich bedarf die durch Literate Programming angeregte Textproduktion aber einer technischen Infrastruktur aus Software, welche die oben genannten Funktionen des „tangeln“ und „weaven“ übernimmt, darüber hinaus Orte der Speicherung und Präsentation. Prinzipiell bieten geeignete Editoren/Programmiersprachen die Möglichkeit, a) einen fertigen datengestützten Text in ein statisches Format (Postscript/PDF) zu

exportieren, dabei können idealerweise b) Abschnitte einzeln adressiert und ausgewählt (=sichtbar gemacht) werden und der genutzte Code etc. so ausgezeichnet werden, dass dieser durch „weave“ in den Fließtext übernommen wird, Resultate von Suchanfragen etc. „eingefroren“ und ausgegeben werden und c) sind Zeitstempel denkbar, welche bei der Überführung in ein statisches Format die Nachvollziehbarkeit etwaiger Algorithmen, Daten und deren Resultate erlauben. Letztlich können so konkret zuordenbare, zitierbare Versionen (zeitlich und inhaltlich) einer dynamischen Publikation erstellt werden. Die Umsetzung „echter“ dynamischer Publikationen, bzw. die Akzeptanz und Institutionalisierung brauchbarer Infrastruktur ist m.E. noch nicht vollumfänglich erfolgt. Im nächsten Abschnitt sollen drei qualitativ unterschiedliche Techniken vorgestellt werden.

Kunth selbst hat, neben der heute eher randständigen Programmiersprache WEB, das Textsatzsystem TeX entwickelt (zu TeX siehe auch: Präsentation Horstmann W, 2.1). Dieses ist in den STM-Fächern – besonders durch die umfangreiche Unterstützung mathematischer Formeln – weit verbreitet und gilt in einigen Wissenschaftsmilieus als Standard zum Verfassen von Manuskripten. TeX bietet zusätzlich zum Umgang mit Formeln die Möglichkeit, Diagramme etc. aus Codeblöcken zu generieren und über (größere) Umwege sogar dynamischen Input zuzulassen. Analog zu den Dokumenteigenschaften von XML (Präsentation Horstmann W, 2.5) sind in TeX-Dateien Inhalt, Struktur und Formatierungsangaben idealerweise getrennt und damit lassen sich beide "Sprachen" relativ einfach konvertieren. Die klassischen Verlagshäuser (und natürlich OA-Dienste) stellen Templates zur Verfügung und akzeptieren in TeX verfasste Manuskripte neben dem klassischen Word-Input. Beispielsweise auch der in der Konsultation erwähnte Webservice authorea.com (Präsentation Horstmann W, 2.1) bietet die dort gehosteten Preprint-Papers zusätzlich zur Webansicht als TeX-Datei zum "selbergenerieren" als Download an. (Aus eigener (Test-)Erfahrung: leider lassen sich aus den LaTeX-Dateien der Authorea-Webseite ohne Vorwissen und mit unveränderten Standardeinstellung der weitverbreiteten TeX-Software (TeX Live) nur rudimentäre oder fehlerhafte PDFs erzeugen.) Mit TeX steht also eine „Auszeichnungssprache“ bereit, welche einerseits klein – im Sinne von: Speicherplatz – , menschen-lesbar und selbststrukturierend ist, andererseits zum Ausführen, d.h. zur Präsentation und Konvertierung, auf eine zusätzliche Infrastruktur an Software angewiesen ist (welche die „Standardausrüstung“ Browser und Adobe Reader übersteigt).
Komplementär zum Textsatzsystem TeX soll an dieser Stelle exemplarisch auf den Texteditor GNU Emacs [https://www.gnu.org/software/emacs/] verwiesen werden. Prinzipiell lassen sich Programme in der Regel in einer simplen Textdatei verfassen und übertragen, bedürfen ihrerseits aber eines sogenannten Interpreters (oder Compilers; letztlich also einer Software welche ein Programm „liest“ und ausführt) um zu „laufen“. Emacs ist ein Texteditor, welcher um diverse

Interpreter (genauer: die Möglichkeit diese zu steuern) modular erweiterbar ist. Damit lassen sich Fließtext und Code verschiedener Programmiersprachen in einem einzigen Dokument (genauer: z.B. in einem sogenannten Org-Mode [https://orgmode.org] Buffer) halten, Codeblöcke einzeln ausführen und deren Resultate zwischen den verschiedenen Blöcken austauschen. Durch Tagging lassen sich Code- und Fließtextblöcke eindeutig adressieren und für einen Export nach HTML oder in eine PDF-Datei ein- und ausschalten. Außerdem können mit Emacs Manuskripte direkt an einen HTML-Server oder beispielsweise eine Versionierungsplattform wie GitHub exportiert werden. (Es muss an dieser Stelle aber erwähnt werden, dass es sich beim Emacs zwar um ein offenes Softwareprodukt handelt, was kostenlos verfügbar und beliebig modifizierbar ist, dessen Einrichtung mit allen genannten Features für Einsteiger oder Nutzer die eine Software „Ouf of the box“ nutzen möchten, kaum zu bewerkstelligen ist. Oder anders formuliert: die Lernkurve ist in der Regel sehr steil.)
Tendenziell dem Literate Programming-Paradigma verpflichtet, ist auch der Ansatz der in „01.05–Grundlagen des Programmierens“ bei Michael Schwab verwendeten Jupiter Notebooks. Da die Notebooks den meisten Kursteilnehmern bekannt sind, soll an dieser Stelle auf ein ähnliches Tool bzw. auf einen Service, nämlich auf bookdown.org (von: *Book* und Mark*down*, einer simplen Markup-Sprache) verwiesen werden. Während Jupiter Notebooks zumeist direkt auf der Plattform GitHub gespeichert und per Link verbreitet werden, bietet bookdown.org eine eigene Veröffentlichungsplattform (sozusagen: ein fachbezogenes R-Repository?) an. Die später bei bookdown.org präsentierten Bücher werden in der Softwareumgebung der Statistiksoftware R erstellt (oder in jedem beliebigen Texteditor), um dann auf die bookdown-Webseite oder einen externen Host (auch hier in der Regel GitHub) hochgeladen zu werden. Die Ansicht erfolgt im Browser und lässt interaktive Elemente im jeweiligen *Book* zu. Beim Export eines Books kann eine ePub-Datei oder statisches PDF erstellt werden, wobei Online noch interaktiv funktionierende Elemente im generierten PDF nicht oder mit Platzhalter versehen und/oder mit Fehlermeldung angezeigt werden. Mit dem Literate Code kann im Sinne von Reuse und Remix gearbeitet werden, indem der auf GitHub hinterlegte Sourcecode eingesehen, heruntergeladen bzw. geforkt wird.

Mit dieser rudimentären und unvollständigen Vorstellung von Tools und Services (rechtliche Aspekte etc. wurden bewusst ausgeklammert) im Kontext von Literate Programming sollte ein kurzer Einblick in die Möglichkeiten und Schwierigkeiten bei der Umsetzung des Konzepts gegeben werden. Hinsichtlich der Übertragung des Paradigmas auf dynamische Publikationen und deren technische Ausführung muss konstatiert werden, dass insbesondere die Darstellung (Präsentierbarkeit), Übertragung und Archivierbarkeit in den Fokus rückt. Für „echten“ dynamischen Content bietet sich die Darstellung im Browser (denkbar sind aber auch Apps oder

andere geschlossene Softwaresysteme) an. Durch Versionierung, direkte Adressierbarkeit und der Erzeugung zeitgestempelter „Snapshots“ lässt sich eine graduelle Archivierbarkeit, Zitierfähigkeit und Authentizität der Ergebnisse realisieren und ggf. ist so auch eine klassische Print-Veröffentlichung realisierbar. Für Bibliotheken steht neben der Schulung in Techniken und Methoden, die Entwicklung und Bereitstellung von Veröffentlichungsinfrastruktur und das Datenmanagement im Vordergrund.

Literatur

[1] Heller L, The R, Bartling S (2014) Dynamic Publication Formats and Collaborative Authoring. in: Bartling S und Friesike S (Hrsg.) Opening Science. Springer 2014

[2] Herb U (2012) Offenheit und wissenschaftliche Werke: Open Access, Open Review, Open Metrics, Open Science & Open Knowledge. in: Herb U (Hrsg.) Open Initiatives: Offenheit in der digitalen Welt und Wissenschaft. universaar 2012

[3] Knuth DE (1984) Literate Programming. The Computer Journal, Volume 27, Issue 2. https://doi.org/10.1093/comjnl/27.2.97

[4] Childs B (1991) Literate Programming, why? Texas A&M University. http://www.literateprogramming.com/bchilds1.pdf (last access: 05.03.2020)