Using Programming Environments for Academic Research and Writing

Morgan Lemmer-Webber[1]*

[1]Art History, Executive Director of the World History Association,
Director of FOSS & Crafts Studios LLC
*Correspondence: morganlemmerwebber@gmail.com
Presented at the ARCHEO.FOSS XV 2021: Open software, hardware, processes, data, and format in archaeological research; on-line; November 23rd-26th 2021.

Abstract: Developer tools, such as code editors, markup languages, and revision control have a greater range of functions than word processors. As a scholar engaged in both Digital Humanities and the FOSS community, I have become increasingly interested in how these tools can be applied to research workflows. I wrote my dissertation in the editor Dr. Racket using Scribble, allowing me to incorporate code directly into my document. In this paper, I discuss the benefits and pitfalls of this decision.

Keywords: Markup languages; Revision control; Digital humanities

FOSS software used and license:
Scribble, MIT license and Apache License v 2.0;
Racket and Dr. Racket, MIT license and the Apache License version 2.0 (some components distributed under the GNU Lesser General Public License, version 3);
git-annex, AGPL version 3 or higher (parts of git-annex are licensed under the GPL, BSD, and other licenses);
GNU Emacs, GPL;
LibreOffice, Mozilla Public License v2.0

Open dataset and license:
N/A

Repository and license:
https://mlemmer.org/git/dissertation/, Apache License v2 (non-Scribble racket files) and Creative Commons Attribution-Share Alike 4.0 International (document files).

Introduction

I have been an active user of free and open source technology for about fifteen years and have run Linux distributions for the operating system on my primary computer for over a decade. For most of that time, my free software advocacy and use has run in parallel with little overlap to my academic studies as an art historian focusing on the art and archaeology of the Roman Empire. However, when I started getting involved in digital humanities projects, I had more tangible reasons to incorporate that technology into my academic projects. This article will discuss my own experiences using programming tools in my research workflow and outline how these methods could be more broadly applied (fig. 1).

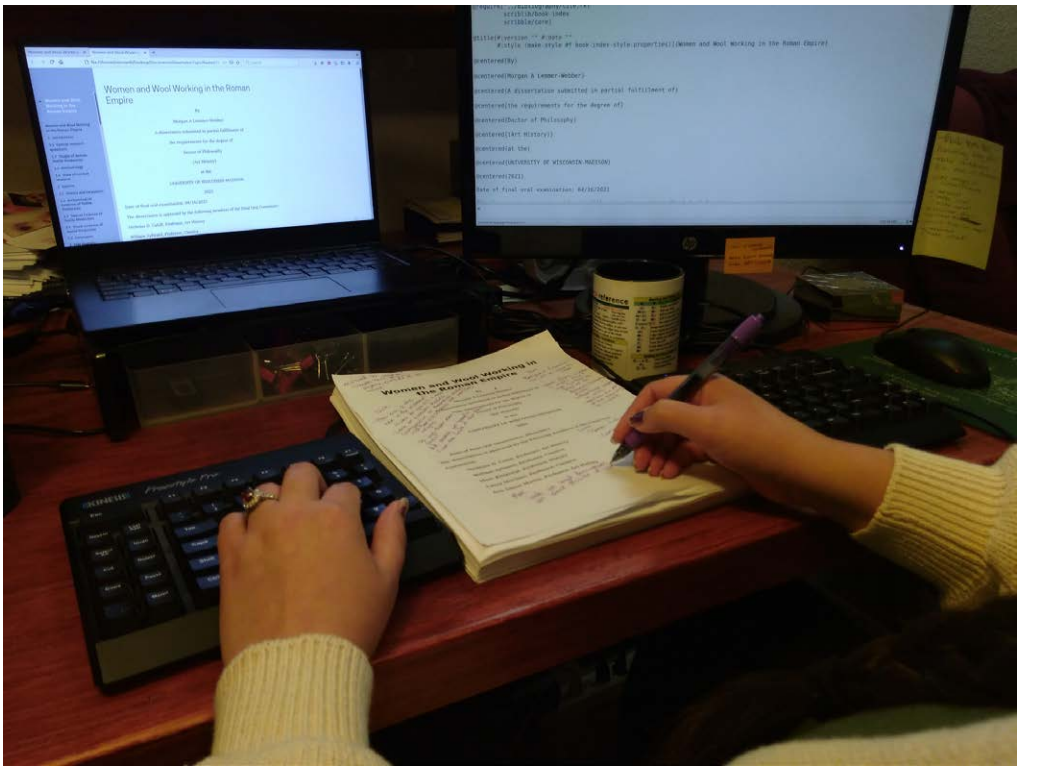

Figure 1. Morgan Lemmer-Webber's dissertation workflow as she incorporates notes written on the physical draft of her dissertation into the scribble sourcecode (right monitor), which is displayed via html in the browser (left monitor).

Digital Humanities Workshops

I co-developed and ran a series of digital humanities workshops with my wife, Christine Lemmer-Webber, to teach the basics of programming to students with no math or computer science background (Lemmer-Webber and Lemmer-Webber 2018). For these workshops, we decided to use the programming language Racket (Racket 2022) because it had a code editor, Dr. Racket, and markup language, Scribble (Scribble 2022), built in. Based on my own experience learning Python, as well as conversations with other humanities students about their anxieties around learning programming, we made a conscious effort to tailor the workshops to minimize those anxieties. The objective of the three-hour workshops was to give a basic introduction to the concepts of programming with a deliverable skill of writing an academic paper using Scribble. Christine wrote a programming tutorial that introduced the basic principles of programming using Racket's picture language to build a snowman. We found that using basic geometric shapes as our universal language eased the fears of some students who thought they wouldn't be able to code because they did not have a math background. Since the number of code projects that are accessible within a basic tutorial that will also have a deliverable outcome relevant to students are fairly limited, I created a tutorial for using Scribble to write an academic paper. This continued the code elements of the workshops using a method that is relatively easy to pick up and had the potential for immediate application.

Dissertation Workflow

Scribble

When it came time to write my dissertation (Lemmer-Webber 2021), I decided to practice what I preach and use Scribble with Dr. Racket as my primary writing environment. This technology provided many advantages over standard word processors. As with any markup language, Scribble made it easier to maintain consistent formatting throughout my dissertation. There was no risk of pasting in a quote from another source and corrupting the formatting for the rest of the section, for example. I was able to export to multiple file-formats including HTML and PDF, from the same source file.

One of the basic features of writing in a code editor that I found most helpful was the ability to comment out text. This means that the text remains in the source document but does not export into the final versions of the text. In programming, this is often used for writing documentation about how your code words or keeping earlier drafts of your code so that you know what you have already tried. When writing academic work, I used this feature to make to-do items for myself, add notes to myself, include feedback from my committee members, or to indicate text that was not immediately relevant but could be useful elsewhere. In a screenshot of my commercial chapter (fig. 2), you can distinguish the sections that are commented out with @; and appear in tan. At the bottom of that page is a note from one of my committee members reminding me to keep the focus on my topic. Near the middle of the page is a table and paragraph which I subsequently commented out to heed that advice. Having the ability to keep all of this text in one location allowed me to remain organized without cluttering the drafts I was sending out to my advisors.

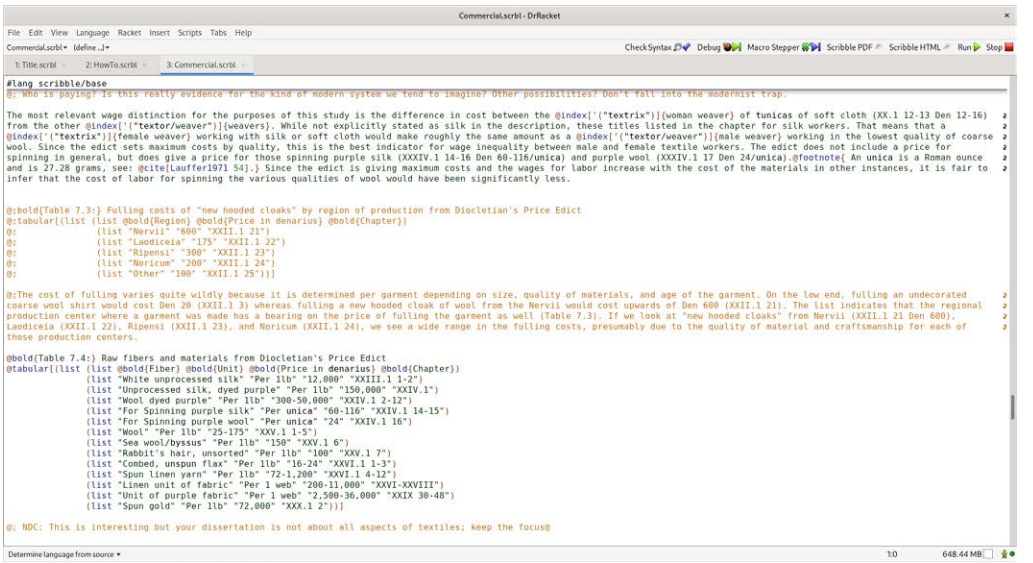

Figure 2. Screenshot of the commercial chapter of my dissertation showing Scribble formatting including commented out notes, and tags for the index, citations, and tables.

Since Scribble is associated with the programming language Racket, I was able to code functions directly into the source document. The document is still effectively Racket code but when you set the programming language within Dr. Racket to `#lang Scribble/doc`, it inverts the typical structure of the programming environment to assume that everything you input is a string unless you call a function with the `@` symbol. Scribble has a wide array of functions out of the box including bibliography, footnotes, images, and figures. However, it was intended for students and researchers within the discipline of computer science and therefore did not have all of the nuances that I needed as a PhD candidate in the humanities.

Luckily, the association with Racket allows for custom code to be built into Scribble documents. Working with Christine Lemmer-Webber we were able to automate certain functions in ways that were relevant to my field. The bibliographic style options for Scribble were limited to those typical of computer science journals and did not have an option appropriate to Art History or Archaeology. The way the bibliography for Scribble operates relies on creating a new function for each citation and manually entering the bibliographic information. However, I also had an existing database of bibliographic metadata for my dissertation references from the citation manager Zotero and wanted to avoid reduplicating work unnecessarily (Zotero 2022). Since we had to re-create the bibliographic system to meet the requirements of my field, we developed a Racket program that compiled the bibliography from the XML database into an output that followed the style guide of the *AJA*. Like the bibliography, the boilerplate structure for creating figures includes manually entering the image descriptions. Since I wanted to be as consistent in the image identifications as possible, we wrote a piece of custom racket code to draw that information from a CSV database and procedurally generated the descriptions.

Scribble has an existing structure for compiling a multi-chapter document. In a standard word processor, this would likely be achieved either by using one large document for the entire dissertation, which can become cumbersome or by creating individual documents for each chapter and then manually pasting them into a full draft in the end stages of the process. In Scribble, you can use the `@include-section` command to append additional Scribble files on one master document. In the case of my dissertation, I used the title page as my master document and appended the individual chapters as well as call other functions that will be procedurally generated. Functionally, this means that exporting the title page compiles the whole dissertation including the Scribble files for individual chapters as well as calls functions to generate the image list, bibliography, and index (fig. 3).

The built-in option to export to PDF is formatted based on conventions in computer science and did not incorporate an easy way to reset the formatting parameters of the output. Formatting LaTeX to a specified style guide is already a difficult feat on its own. Formatting generated LaTeX from a markup language adds a further layer of difficulty. Ultimately, we were only able to get about 90% of the formatting to output to PDF correctly on export. In the end, Christine built a separate export option for the Open Document Format or ODF (Durusau and Svante 2021). Once exported to ODF, I was able to open the document and make the final small adjustments using the open source word processor LibreOffice (The Document Foundation 2022).

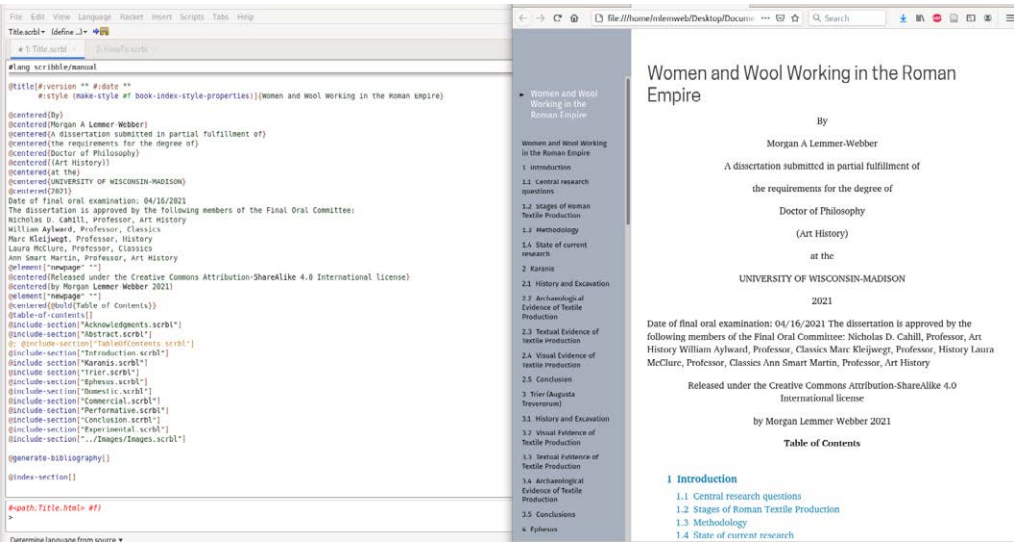

Figure 3. Screenshots of the title page of my dissertation (a) in the Scribble format this serves as the master document with the separate sections included (b) The HTML output of my dissertation with the title page visible and the hyperlinked table of contents both below and to the left of the title page.

Revision Control

Since I was already using programming tools for my dissertation, I further enhanced my workflow by using revision control. This process saves changes to a file incrementally over time, effectively allowing the user a time-traveling view of document history. For many academic writers, saving incremental or even periodic changes requires saving each version of the document with a slightly different file name. This clutters your file system and can be difficult to navigate to find the specific change you are searching for. Git, the most common free and open source system for revision control, saves these versions in a separate repository, leaving only one copy of the document in your file system and prompts you to explain what changes you've made in a commit message (Git 2022). This is not a feature built into Dr. Racket, nor most standard word processors. It is most typically accessed either through the command line, or through a platform such as GitHub that offers a graphical user interface. Git can be used with most text-based file types, so revision control is accessible if you are writing using a standard word processor. Since Git on its own does not handle images well, I added Git Annex, a free software revision control system built for large file types, in order to include all of the files necessary to export my dissertation in the same repository (Git-annex 2022).

Access to Git repositories can be shared with collaborators and track who made changes to a document and when. This makes them indispensable for team projects. While co-developing the custom Racket functions for my dissertation it was necessary to commit the code regularly so that Christine and I were both working with the most recent version. It is worth noting that this is a very foreign workflow for most people and the command line interface is notoriously difficult even for many experienced developers, let alone a humanities student. The majority of work on a dissertation is done alone. When not coding on a regular basis or having another person reliant on the most recent update, it is much easier to forget to commit your changes

even if it is still good practice. In order to compensate for this user oversight, I used a program called Git Annex Assistant to automatically commit my dissertation files to Git and Git Annex periodically (Git-annex/assistant 2022).

Obstacles

Unfortunately, achieving these benefits came with its own share of obstacles. While Scribble on its own has a relatively low barrier to entry for the boilerplate options, it is still higher than standard word processors. All markup languages are an adjustment, particularly for users who are acclimated to graphical user interfaces for formatting. With few exceptions, making a single error in formatting in a standard word processor is unlikely to crash your entire document. When using a markup language, a single missed closing parentheses or a bracket instead of a curly bracket will result in an error message and the inability to export your document until it is fixed. The interface of Dr. Racket does syntax highlighting on functions to indicate which information is covered within that function and the line where the error occurs will be highlighted in pink on the right side of the screen. These indicators help locate such bugs as they occur and are reasonably easy to identify if you are mindful of them as you work. The real issues arise when you do not catch a bug soon enough, particularly if you have multiple bugs at once. The best preventative measure against this, in my experience, is to export the document to HTML once per paragraph written. This way you have a constrained area of text to search for bugs if the export fails making it more manageable than searching through pages of text.

While novice programmers like myself can absolutely debug Scribble errors, define new functions and even incorporate simple code into their source documents, for the higher-level functions and customization I did have to rely on assistance from a more experienced developer. Without access to such guidance this process would have been stalled and I would have either been limited to the boilerplate options available in Scribble or given up and returned to LibreOffice. Before Scribble can be widely adopted by non-programmers, I believe more infrastructural work needs to be laid out to make the options more adaptable to the needs of other fields.

Incorporating git and git-annex as someone with minimal development experience likewise had a steep learning curve. The command line is not an intuitive interface even for many experienced programmers. For people who have minimal command line experience, committing to git is more labor and time investment than it is to programmers who commit code several times daily. Programmers are also more likely to have the command line open as they test their code, run virtual environments, or perform other routine tasks. When your daily work does not require the command line, it is difficult to remember to commit regularly, thus the necessity for a program to auto-commit my dissertation. This workaround was not without flaws, particularly since the auto-commits do not include commit messages. I, therefore, sacrificed having an easily navigable repository with messages explaining what changes were but with those commits happening for a repository with regular commits in short intervals that are likely to catch more changes but without context clues as to what those changes were.

While writing my dissertation in Scribble had many benefits, it also added layers of stress to an already difficult situation. Like most people within academia, I had years of experience

writing papers using a word processor. While a lot of the tasks in that workflow are tedious, they are at least familiar. Using a markup language introduced unexpected issues that I wasn't always able to fix in a timely manner. It is difficult explaining to your committee members that while italicizing words in foreign languages is a trivial task in a word processor, it was an export error for my dissertation. Even though none of the text in the exported draft that I submitted had italics, behind the scenes the correct words were still marked as italic in the source document but a solution we applied to fix one formatting error somehow corrupted other areas of formatting. This issue arose because debugging LaTeX output is difficult at the best of times but debugging auto-generated Latex as an intermediate step between the markup language (Scribble) and the final document format (PDF) is even more difficult. The level of complexity here can be inferred by the fact that it was easier, in the end, to write an entirely new export method to ODF than debug the existing Scribble to LaTeX to PDF exporter. With a lot of work, we were able to find workarounds for all of these issues by the time I needed to submit my dissertation but troubleshooting these issues averted my attention away from writing the content of my dissertation. That being said, this article is an overview of my own personal experience using these technologies and therefore has a sample set of one. I have only written one dissertation, and without a control sample, it is hard to say whether I spent more time trying to debug my encoded dissertation than I would have individually formatting every image, manually compiling an index and all of the other minutiae that were successfully automated in my workflow.

Gnu Emacs

When I started working on my dissertation, Dr. Racket and Scribble were the obvious choices of platform because they were the most familiar programming environment to me. I knew that there were more powerful programming environments out there but did not anticipate the number of obstacles that would arise from scaling the tools I knew from a seminar paper to a dissertation. We realized midway through the process that using a more robust editor such as Gnu Emacs would have eliminated some of the more difficult obstacles (Gnu Emacs 2022). However, at that point, my dissertation deadlines were looming too close to completely change my workflow. I began learning Emacs after I submitted my dissertation and had more free time to invest.

The Emacs programming environment has been consistently used and developed since it appeared on the scene in 1976. This means that it has accrued a powerful assortment of features that can adapt to most computing needs (Lemmer-Webber and Lemmer-Webber 2022). This long history, however, also creates its own barriers. The keyboard shortcuts for Emacs were developed before the standardized shortcuts we all know today. Instead of using Ctrl-c and Ctrl-v to 'copy' and 'paste', for example, you use Ctrl-w and Ctrl-y to 'kill' and 'yank' (Free Software Foundation 2016). It has gone through various iterations and I am using Gnu Emacs specifically. While the program does have a graphical user interface, it was designed to be navigated using keyboard shortcuts and therefore is less intuitive than many other programs.

Whereas Dr. Racket was created as a programming environment purpose-built for Racket and Racket-based languages, Emacs has support for most standard and many obscure programming languages. Therefore you are similarly able to write custom code into your source document, but you are able to choose your programming language of choice (i.e. python, R, Racket). This versatility expands to markup languages as well and Emacs is compatible with markdown,

LaTeX, and HTML, among others and can export to PDF, HTML, LaTeX and ODT. When working on the Scribble ODF exporter, Christine reverse-engineered and adapted the Emacs ODF exporter as a template.

Emacs-Org Mode is an organizational system that uses a simple but versatile markup system that can be used for outlining, task assignment, project planning, and to write text documents (fig. 4). Since I am currently not writing code on a regular basis, the majority of my tasks in Emacs have been using either Org Mode or markdown. These tasks vary but include documents to track my job application process, write updates and manage my personal website, write tutorials and educational materials, collaborate on outlines for projects, and manage contracting clients.

The other main aspect of my workflow that Emacs has tremendously improved is Magit, a git porcelain that is integrated into the Emacs interface (Magit 2022). This means that I can access a much more intuitive interface for git that is accessible through the same program I am writing in and doesn't require a separate terminal or command line interface (fig. 4). It is far easier to remember to stage and commit changes to a document when it only requires a handful of keystrokes and a commit message rather than switching interfaces entirely.

Given the nature of this article as a retrospective of my experiences, it felt fitting to write the article itself in Emacs to expand that experiential aspect. While this brief writing exercise is far less complex than a dissertation as a whole, it has been an interesting comparison. The main limitation to this workflow in this particular instance is that instead of a style guide for the formatting, the editorial board of the ArcheoFOSS conference provided a .docx template with formatting built in. This means that I am writing the document in Org Mode using Emacs, exporting it to ODF, then assembling it into the template in LibreOffice.

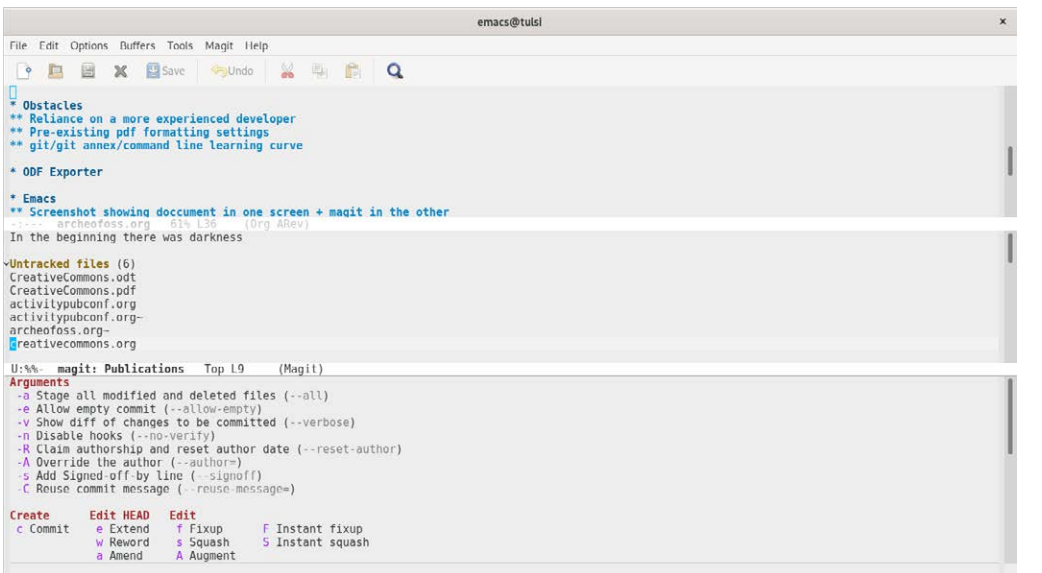

Figure 4. A screenshot of the outline for this article in Emacs Org Mode visible at the top of the screen, and the Magit interface to commit the document to Git at the bottom of the screen.

Conclusions

My experience using Scribble to write my dissertation was unique and, in many ways, experimental. There were learning curves along the way that exceeded most dissertation experiences. I believe that incorporating these types of programming tools into my research workflow has greatly enhanced my experience. I was able to hone skills that were applicable both to my academic career and more broadly marketable while minimizing the list of tedious tasks that typically overwhelm students at the end stages of the dissertation process. If I were to go back to the start, I would still begin the learning process with Dr. Racket and Scribble. Emacs itself is too overwhelming and its user interface too foreign to be a useful entry point to using programming environments. Having an intermediate stage that has a comparatively low barrier to entry allowed me to gain the confidence I needed to later take on learning Emacs. Though, given the limitations we hit with Scribble for a complex document with strict formatting regulations with my dissertation, I would advise anyone who is contemplating this approach to take the time to learn a more robust code editor prior to beginning a dissertation- or book-length project.

I believe that incorporating these types of programming tools into research workflows has significant merit and broad application. While the custom features written into my dissertation were primarily intended to automate the processes, which are tedious and time-consuming, the ability to write custom code into a source document has an infinite number of applications for projects whose needs exceed standard word processors. Projects that involve large amounts of data analysis, for example, could write these functions directly into their source file rather than compiling the information elsewhere and then incorporating it. A project that compiles data from a changing or fluctuating pool of information, such as annual reports for an excavation, could create a custom template that auto-generates the annual statistics into a consistent format. Any project which requires multiple users to edit a document could benefit from revision control which monitors who made changes, when, and preserves older copies of the document in the event that an error is made. Furthermore, having a publicly available git repository where the data, source code, methods, and reports are available greatly increases reproducibility. While digital tools and processes have increasingly revolutionized the way that data and research is interpreted, visualized, and shared, the writing workflow for many scholars has remained relatively unchanged. Imagine how much more we could achieve if we think outside of the .docx.

Supplementary Materials: As the hosts of the podcast *FOSS and Crafts*, Christine and I have recorded episodes about many of the themes covered in this article. The following are available online at https://fossandcrafts.org, Podcast S1: Digital Humanities Workshops, Podcast S2: Scribble and the Open Document Format, Podcast S3: Learning Emacs.

Funding: This research received no external funding.

Acknowledgments: I am deeply indebted to my wife, Christine Lemmer-Webber, for her assistance in developing the custom code embedded into my dissertation as well as her work co-developing the digital humanities workshops and the podcast FOSS and Crafts.

References

The Document Foundation. 2022. "LibreOffice." Accessed February 28, 2022. https://www.libreoffice.org/.

Durusau, P. and S.Schubert (eds). 2021. "OASIS Open Document Format for Office Applications (OpenDocument) TC v1.3." Accessed February 28, 2022. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office.

Free Software Foundation. 2016. "GNU Emacs manual." Accessed February 28, 2022. https://www.gnu.org/software/emacs/manual/emacs.html.

"Git – fast-version-control." 2022. Accessed February 28, 2022. https://git-scm.com/.

"Git-annex." 2022. Accessed February 27, 2022. https://git-annex.branchable.com/.

"Git-annex/assistant." 2022. Accessed February 27, 2022. https://git-annex.branchable.com/assistant/.

"Gnu Emacs." 2022. Accessed February 27, 2022. https://www.gnu.org/software/emacs/.

Lemmer-Webber, C. and M. Lemmer-Webber. 2020. "Scribble and the Open Document Format" Podcast episode. *FOSS and Crafts,* November 5, 2020. https://fossandcrafts.org/episodes/15-Scribble-and-the-open-document-format.html.

Lemmer-Webber, C. and M. Lemmer-Webber. 2022. "Learning Emacs." Podcast episode. *FOSS and Crafts,* February 5, 2022. https://fossandcrafts.org/episodes/41-learning-emacs.html.

Lemmer-Webber, M.. 2021. "Women and Wool Working in the Roman Empire." Doctoral Dissertation, University of Wisconsin Madison. https://mlemmer.org/dissertation/

"Magit, A Git Porcelain inside Emacs." 2022. Accessed February 27, 2022. https://magit.vc/.

"Racket, the Programming Language." 2022. Accessed February 27, 2022, https://racket-lang.org/.

"Scribble: The Racket Documentation Tool." 2022. Accessed February 27, 2022. https://docs.racket-lang.org/Scribble/.

"Zotero, Your personal research assistant." 2022. Accessed February 27, 2022. https://www.zotero.org/.