DHSI

DIGITAL HUMANITIES SUMMER INSTITUTE

Creating Digital Collections with Minimal Infrastructure: Hands On with CollectionBuilder for Teaching and Exhibits

Olivia Wikle, Evan Williamson, and Devin Becker

ETCL
ELECTRONIC TEXTUAL CULTURES LAB

Social Sciences and Humanities Research Council of Canada
Conseil de recherches en sciences humaines du Canada

Welcome to DHSI 2023!

Thank you for joining the DHSI community!

In this coursepack, you will find essential workshop materials prefaced by some useful general information about DHSI 2023.

Given our community's focus on things computational, it will be a surprise to no one that we might expect additional information and materials online for some of the workshops—which will be made available to you where applicable—or that the most current version of all DHSI-related information may be found on our website at dhsi.org. Do check in there first if you need any information that's not in this coursepack.

Please also note that materials in DHSI's online workshop folders could be updated at any point. We recommend checking back on any DHSI online workshop folder(s) that have been shared with you in case additional materials are added as DHSI approaches and takes place.

And please don't hesitate to be in touch with us at **institut@uvic.ca** or via Twitter at **@AlyssaA_DHSI** or **@DHInstitute** if we can be of any help.

We hope you enjoy your time with us!

DHSI
DIGITAL HUMANITIES SUMMER INSTITUTE

ETCL
ELECTRONIC TEXTUAL CULTURES LAB

Statement of Ethics & Inclusion

Please review the DHSI Statement of Ethics & Inclusion available here: **https://dhsi.org/statement-of-ethics-inclusion/**

DHSI is dedicated to offering a safe, respectful, friendly, and collegial environment for the benefit of everyone who attends and for the advancement of the interests that bring us together. There is no place at DHSI for harassment or intimidation of any kind.

By registering for DHSI, you have agreed to comply with these commitments.

Virtual Sessions

Your registration in DHSI 2023 also includes access to the virtual **institute lecture** sessions. Access details for these talks will be shared as DHSI approaches.

Due to the high volume of attendees, please ensure your DHSI registration name or DHSI preferred name and your Zoom name match so that we know to let you into the virtual sessions.

DHSI Materials

DHSI materials (ex. videos, documents, etc.) are intended for registrant use only. By registering, you have agreed that you will not circulate any DHSI content. If someone asks you for the materials, please invite them to complete the registration form to request access or contact us at **institut@uvic.ca**.

Auditor and participant registration

If you registered to **audit** any workshops, note that auditor involvement is intended to be fully self-directed without active participation in the workshop. The auditor option offers more flexibility regarding pace and time with the workshop content. Your registration as an auditor will include access to some asynchronous workshop materials only and does not include access to live workshop sessions and/or individual/group instruction or consultation. Please direct any questions about DHSI workshop auditing to **institut@uvic.ca**.

If you registered as a **participant** in any workshops, your registration includes access to asynchronous content + active participation in live workshop session(s). The workshop instructor(s) will contact you about the date(s), time(s), and platform(s) of the live workshop session(s).

If you are unsure whether you registered as an auditor or participant, please check your registration confirmation email. Further questions can be directed to **institut@uvic.ca**.

Schedule

The at-a-glance schedule of DHSI 2023 courses, workshops, institute lectures and aligned conferences & events can be found here:
https://dhsi.org/timetable/

All times are listed in North American **Pacific Time Zone**.

For those who registered as participants in any workshops, live sessions for online workshops are not currently listed on the above-referenced schedule. **Instructors will be in touch with registered participants directly about the exact date(s) and time(s) of their live workshop session(s).**

Acknowledgements

We would like to thank our partners and sponsors (including the Social Sciences and Humanities Research Council), workshop instructors, aligned conference & event organizers, institute lecturers, local facilitators, and beyond for making this possible.

Further information

General DHSI 2023 information: **https://dhsi.org/program/**

Full course listings (in-person): **https://dhsi.org/on-campus-courses/**

Full workshop listings (online): **https://dhsi.org/online-workshops/**

Aligned conferences & events (in-person): **https://dhsi.org/on-campus-aligned-conferences-events/**

Aligned conferences & events (online): **https://dhsi.org/online-aligned-conferences-events/**

Institute lectures: **https://dhsi.org/institute-lectures/**

Frequently asked questions: **https://dhsi.org/faq/**

Any questions not addressed in the above pages? Please email us at **institut@uvic.ca**!

DHSI 2023 Course Pack

Creating Digital Collections with Minimal Infrastructure: Hands On With CollectionBuilder for Teaching and Exhibits.

This document will change as we progress through the week adjusting to the progress and interests of the group! The living version of this document can be found here: https://is.gd/cb_dhsi2023

Dates: **5-9 June 2022**
Class Time: sessions follow the official DHSI timetable.
Location: University of Victoria

Instructors:

- Olivia Wikle (omwikle@uidaho.edu)
- Evan Williamson (ewilliamson@uidaho.edu)
- Devin Becker (dbecker@uidaho.edu)

Course Description

This course introduces fundamental web and DH skills using CollectionBuilder, an open source project for building digital collection and exhibit websites driven by metadata and hosted on a lightweight infrastructure.

The high cost and IT requirements of digital collection platforms are often a barrier to creating new collections for sharing or teaching humanities research. CollectionBuilder is optimized for non-developers and simple hosting solutions, allowing researchers to take greater ownership over their digital projects and lowering barriers to customization.

Scholars in this course will learn CollectionBuilder by engaging in a scaffolded approach with hands-on experience in digital library foundations such as scanning and metadata creation to web development. Building on these skills, students will learn the basics of working with plain text files, CSV data, Markdown, Jekyll, Git, GitHub, and GitHub Pages in order to create and customize their very own digital collection. By the end of this course, students will have gained the knowledge and independence necessary to implement CollectionBuilder in contexts that include creating and disseminating research collections and custom digital exhibits, or teaching digital libraries in the classroom.
This is a hands-on course that will cover basics of digitization, metadata, and web programming fundamentals. No programming experience is necessary, although you should have a strong interest to learn! Participants are asked to bring their own computers. All software used in the course is free, open

source, and cross platform and will be installed during class time. Optionally, participants are invited to bring along a small collection of physical items to digitize, digital files (images, pdfs, audio) to feature in a digital collection, or metadata exported from an existing collection hosted on CONTENTdm.

Code of Conduct

We are committed to creating a respectful learning environment that is inclusive of participants with all backgrounds and abilities. This course will adhere to the DHSI Statement of Ethics & Inclusion, and the CollectionBuilder Code of Conduct.

By engaging with the methods of creating digital collections that will be introduced in this course, participants will become valued members of the CollectionBuilder community, a primary goal of which is to be inclusive to contributors with varied and diverse backgrounds. As community participants, we pledge to make participation in our project and community a respectful and harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We invite all those who participate in the CollectionBuilder project and community to help us create safe and positive experiences for everyone.

Course Objectives

By the end of the course, we hope you will:

- Understand how static website generators work and what advantages and disadvantages they offer.
- Be familiar with the base functions and customizable options that CollectionBuilder offers through its various templates.
- Be comfortable using a development environment to build static sites.
- Build one customized and polished digital collection and understand how and where you could publish the collection online.
- Be able to teach or introduce CollectionBuilder and Jekyll to others.

Course Software & Data

One of the more challenging parts of working with data and the web is getting a development environment set up that you understand and that makes your work easier. We believe the environment we use to develop with CollectionBuilder is pretty good, but even amongst the three of us we disagree on some of the components. This week we will be teaching you how to set up a development environment like ours, but the most important part is that you feel comfortable and enabled by the tools and software you use to build the sites. Below are links and short details to the tools/software we use to develop CollectionBuilder.

Please use our CollectionBuilder documentation for help with installing all these pieces: https://collectionbuilder.github.io/cb-docs/docs/software/

Spreadsheets for Metadata Management

With CollectionBuilder, Everything starts with metadata, so it's important to have reliable means for working on, collaborating, and transforming your data. We use Google Sheets because we find that it's the easiest platform to collaborate on and, unlike Excel, it does less automatic transformations on one's data. Google sheets also facilitates an easy download of the data into a CSV format. Collection data must be in a CSV format for CollectionBuilder to process it, and we've found Sheets is reliable for this (Excel can not save a correctly formatted CSV).

For more complicated projects, we also use Open Refine, which is excellent with data transformations. But for this class and this level of project, Google Sheets should be fine. If you would prefer a desktop alternative, LibreOffice Calc is a good free and open source spreadsheet program that will handle encodings and CSVs correctly.

Text Editor - Visual Studio Code

We use Visual Studio Code, an integrated development environment (IDE). VS Code is a free, open source Microsoft product, and quite popular with developers worldwide. VS Code allows for easy access to the project's file structure, as well as a terminal and text editor all in one screen. Other text editors (Atom, for instance) will work fine as well, but we're most familiar with this one and have found it to be user-friendly and increasingly powerful as one gets better acquainted with its many features and plugins. One plugin we'll have you download right away is Rainbow CSV, which adds color cues to CSV files that are displayed in the system.

Version Control Software (Git + GitHub)

Git is the most popular version control software, and is free and open source. We use Git to track the development of our projects and to assist us with collaborating. It's important to note that Git is

different than GitHub, which is a platform that uses Git to enable collaborative software development. Git is the software that enables one to track differences in file versions. GitHub is a platform that uses Git to enable collaborative development and online publication of the git process and web outputs. We use GitHub, but one could use GitLab, BitBucket, or other repository hosts in a similar way.

We will show you how to use Git and GitHub on the web interface and on your local computer. Locally, some find working through VS Code's built-in version control features helpful, and some of us like using GitHub Desktop to manage the various git commands and tasks. We'll show you both.

Static Site Generator (Jekyll, a generator written in Ruby)

Jekyll is an open source static site generator that runs on the Ruby programming language. Overall, Jekyll has been kind to us, but Ruby is really annoying. (Just kidding, Ruby, you're great!). Really, once you have Ruby set up correctly, there are few problems. But sometimes getting the right version of Ruby set up and correctly configured can be a pain. We have specific how-to instructions for setting up Ruby on Windows, Mac, and Linux. We are also happy to help you during or prior to the class starting (we'd love it if you could get this installed before we start, but we know that might be hard for some!).

As for Jekyll, we gave a lot of thought to what static site generator to use for CollectionBuilder, as there are many site generators available. We came to use Jekyll because the way it represents its code, files and folders made the most sense to us and because it is the most popular version. Here is what we wrote in one of our articles detailing our approach:

> University of Idaho librarians evaluated a wide variety of static site generators, eventually settling on Jekyll for a variety of reasons. First, Jekyll is set up so it supports a simple mental model of how the site will be built that matches up with traditional web development approaches. Static assets in a folder in the source code will become static assets in the same location on the built-out site. Content is represented by stub files that are assigned a layout that pulls together the modular template elements of each web page. This arrangement is similar to the library's earlier templates of PHP includes, built into a tool that makes the approach considerably more powerful and sustainable. University of Idaho librarians' experiences teaching others during classes, workshops, and internal sessions suggest that the biggest barrier to getting started with Jekyll is setting up the development environment, including Ruby, the programming language necessary to run it. Once past that initial hurdle, learners without a development background are able to understand how the tool works and web pages are constructed. In contrast, some of the major alternatives, such as Hugo, GatsbyJS, and Next.JS, seem to rely on a more formal computational mental model for constructing sites, making them amenable to JavaScript developers, but less intuitive to an average librarian.
>
> Second, Liquid, the templating language used by Jekyll, is powerful yet easy to learn, opening new possibilities for driving content generation from simple data formats such as CSV. This

> ability to use data created and edited in spreadsheet formats, allows rethinking much of the website content as re-usable chunks added into pages using flexible templates. Spreadsheets are something library folks have plenty of experience with, providing an easy entry point for collaborators to create, organize, and maintain content on the site.
>
> Finally, Jekyll has become the most popular out of the myriad of emerging static generators. This is in part due to being integrated into GitHub's free web hosting service, GitHub Pages, making it an attractive option for quick projects and learning opportunities. The vibrant community around these tools results in better support when encountering issues and a wide ecosystem of quality examples to draw from.
>
> On the surface, “popularity” might seem like a shallow metric to consider when selecting tools, but it has become a significant factor when evaluating the sustainability and usability of different technology choices. In the library’s context, ready availability of quality documentation and help resources can lower the barriers for learning and use. Additionally, tools such as Jekyll, Bootstrap, and GitHub have huge novice user communities that ask questions and post answers across the web. A quick, specific search will almost always return solutions that are comprehensible to non-computer scientists for any issue one encounters. This accessibility of help resources and a community of users is essential to fostering a library-centric approach as well as keeping the workflow "do-able" for University of Idaho librarians and, the authors argue, for librarians generally.

So yeah, we've given it some thought!

Again, we know getting these environments set up and making them usable is often challenging. One of our main goals for this class is getting you comfortable using this method of development, so please reach out to us with any issues or problems you might encounter.

CollectionBuilder Course Schedule:

By course's end, participants will have created a digital collection with their own material, learned various possibilities for customizing and extending that collection, and discovered the opportunities that CollectionBuilder can provide as a starting point for future DH web development.

Day 0: Preparation

1. Download and install software (Visual Studio Code, Git, GitHub Desktop, Ruby, Jekyll)
    a. Please use our CollectionBuilder documentation for detailed help with installing all these pieces: https://collectionbuilder.github.io/cb-docs/docs/software/
    b. All software is free, open source, and cross platform.
    c. If you run into problems installing some of this software, don't despair! You're welcome to contact us at any point leading up to the workshop and we'll help you troubleshoot. We'll also make time during the workshop week to address these issues, so if you don't get them all installed by Day 1, that's okay.
2. If you don't already have one, create a GitHub account and make sure you'll be able to access it during class time.
3. This class will involve some editing in Google Sheets, so please be sure to have a working Google Drive account set up before the workshop begins. (If you do not use Google products, please have LibreOffice Calc available to edit spreadsheets - Excel does not work for this!)
4. Watch the short introduction to CollectionBuilder video and take some time to explore the CollectionBuilder-GH demo collection.
5. Gather objects for your digital collection
    a. **Supported formats**: jpg, png, pdf, mp3 – plus YouTube, Vimeo, and external links to objects hosted elsewhere
    b. **File size**: the full size object can be any size you think your users might want to download. This might not be your full sized preservation file–generally, we try to provide very high quality objects to users, but balance that against the practicality of huge file sizes–most users don't want a 1GB tif or pdf!
    c. **Filenaming**: to avoid issues, please pay close attention to filenaming conventions! The filename should be:
        i. all lowercase
        ii. no spaces
        iii. no special characters (underscores (_) are okay.
    d. See CollectionBuilder-CSV Objects for more information
6. Copy metadata spreadsheet
    a. Log in to Google Drive. Then, using Google Sheets, make a copy of the CollectionBuilder-CSV metadata template (Click the blue "Make a Copy" button on the page that opens when you click the link).
    b. If you have time before class, we recommend filling out the title and description fields, and adding your object filenames or object links to the object_location column. We

will go over technical metadata fields more in-depth and have metadata work time in class, so there is no need to fill out this entire spreadsheet beforehand. It is also possible (and even common!) to transform existing metadata in other formats and templates into a version for your CollectionBuilder project (e.g. metadata from a different repository platform, finding aid, curated from another exhibit, or project type). We can help you with the transformation during course time.

c. See our metadata guidelines for additional information. Any custom field can be added based on the needs of your project, but column names should be lowercase with no spaces or odd characters.

Day 1: Monday, June 5 - CollectionBuilder-GH and GitHub

Topics: CollectionBuilder intro, GitHub, working in GitHub web interface, build a CollectionBuilder-GH walkthrough

Major Learning Objectives:

Conceptual

- *Have a basic understanding of the structure and design of a CollectionBuilder site*
- *Recognize the required metadata fields for a CollectionBuilder-GH instance*
- *Understand how to find the CollectionBuilder docs and how to use them*

Technical

- *Be able to start a CollectionBuilder project using the CollectionBuilder-GH Template*
- *Understand the basics of working with GitHub web interface*
- *Be able to recognize a .YML file and where to find a _config.yml file in a Jekyll project*

Day 1 Outline:

9:00am - 10:15am - Central DHSI Welcome Session

10:30am - 12:00pm - Course Welcome and Introductions

1. Introductions (instructors and participants)
    a. Name, something about yourself, what are technical things are you afraid of, a favorite digital project
2. Course overview
3. CollectionBuilder introduction - slides (Devin)
    a. Tour a CollectionBuilder demo site
4. Intro to CollectionBuilder docs (Olivia)
    a. Documentation sections
    b. Documentation search

    c. Discussion forum
5. Walkthrough creating a demo CollectionBuilder-GH project - using prepared collection data, we will create demo websites to introduce the basic processes and components of CollectionBuilder.
    a. Overview of GitHub web interface (repositories, issues, etc)
    b. Copy CollectionBuilder-GH Template to create your new project
        i. "use this template" button
        ii. How to name repository projects
        iii. Repository settings intro (where to delete later if you want!)
        iv. Turn on GitHub Pages
    c. Edit the README
        i. Using the GitHub web editor
        ii. Making a commit
        iii. Look at repository history
    d. Download demo data
        i. Psychiana Collection Demo Data
            1. Metadata: https://docs.google.com/spreadsheets/d/1x48Te3duPAxh53foEihQVKTfCKUjaCCbH7TrMMd_yU4/copy
            2. Objects: http://lib.uidaho.edu/collectionbuilder/demo-objects.zip
        ii. Carleton Watkins Mine Collection Demo Data
            1. Metadata: https://docs.google.com/spreadsheets/d/1mThECwBYaUdvUrSbc9d2wbjedpYyvVD89jJ15R-7Qmo/copy
            2. Objects: http://lib.uidaho.edu/collectionbuilder/watkins.zip
    e. Upload Objects
        i. Objects concepts (Evan)
            1. Object types – in CB-GH: jpg (and other images), pdf, and mp3 in the "objects" folder, within GitHub and reasonable web size limits. External items: YouTube, Vimeo, files hosted elsewhere, links – external items are a really powerful option!
            2. File extensions – be sure to turn on displaying file extensions in your file explorer.
            3. File naming conventions – get your objects organized and normalized!
            4. Issues – everything needs to match! Case matters!
        ii. Upload and Commit changes
    f. Upload Metadata
        i. Metadata concepts (Devin)
            1. Why use Google Sheets? No EXCEL!!
            2. Required fields
            3. Formats, filenames, field names

4. OpenRefine is a great tool for wrangling existing metadata–check Getting Started with OpenRefine workshop video for info.

ii. Upload and Commit changes

g. Configure site-wide settings using "_config.yml"
   i. Introduction to YAML
   ii. Edit YAML and Commit changes

h. Look at GitHub repo history

i. Break the demo! Mess around with metadata to see what sort of odd things can happen

1:00pm - 2:30pm - Working with CollectionBuilder-GH demo collections

6. Theme.yml
   a. Add a feature image
7. Config - csvs
   a. Customize nav
   b. Customize browse cards
8. Discussion
   a. Questions?
   b. Discuss how CollectionBuilder-GH has been used for teaching and prototyping ideas.
   c. Create an Issue in the dhsi-demo repository introducing yourself and link to your repository.
9. Software prep
   a. Ensure software is set up on your personal computer and up to date (you may have already done this by now)!
      i. Git
      ii. Text editor
      iii. Ruby
      iv. Jekyll + Bundler
      v. ImageMagick and Ghostscript (if you want to process images and PDFs)

2:45pm - 4:00pm - Institute Lecture

Day 2: Tuesday, June 6 - CollectionBuilder-CSV and Local Development

Topics: Git, Git Clone, Local Development with Jekyll, YML, CSV

Major Learning Objectives:

Conceptual

- *Understand the differences between Git, GitHub, GitHub Pages, and GitHub Desktop*

- *Understand the difference between Ruby and Jekyll*
- *Understand how and why one would develop locally and collaborate via the cloud*
- *Understand the importance of case in the naming of files and file extensions*
- *Understand difference between CollectionBuilder-GH and CollectionBuilder-CSV*

Technical

- *Be able to start a development server on your computer*
- *Be able to locate your project repository and edit your repository files on your computer*
- *Be able to perform the basic Git workflow (commit, push, pull) locally and check results on GitHub + GitHub Pages*
- *Be able to use Google Sheet formulas and filters to fill in your metadata*
- *Be able to edit and add to your collection's metadata and objects*
- *Be able to use _data/theme.yml to customize the basics of your project*
- *Be able to customize your collection's pages using the config files available.*

Day 2 Outline:

9:00am - 10:15am - Check in and discuss individual projects

1. Morning check in
    a. Look at Issue introductions
    b. Introduce Pull Requests
2. CollectionBuilder review and technical overview - slides (Evan)
    a. Define Git, GitHub, GitHub Pages, Jekyll, Ruby, Bundler, CollectionBuilder
3. Discuss personal collection project
    a. What type of items do you have?
    b. Where does your metadata come from?
4. Set up new CollectionBuilder-CSV project for your personal collection on GitHub
    a. Introduction to CB-CSV
    b. Differences from CB-GH
    c. Create "use this template" button
    d. Tips about naming your repository

10:30am - 12:00pm - Set up Local Development

5. From Clone To Push: A CollectionBuilder + GitHub Desktop Step by Step (Devin)
    a. Introduction to Git
        i. Clone your collection to your computer using GitHub Desktop
        ii. Terminology overview:
            1. Repository, local, remote
            2. Git clone
            3. Git add
            4. Git commit
            5. Git push
            6. Git pull

    b. Introduce the local development environment:
        i. Folder of files = CollectionBuilder project
        ii. Text editor: Visual Studio Code
            1. VS Code overview
            2. VS Code features
            3. VS Code plugins
        iii. Command Line Jekyll (bundler best practices/troubleshooting)
            1. "bundle install" (first time only)
            2. "bundle exec jekyll s"
    c. Use Jekyll to serve your site locally
    d. Edit README and walkthrough Git commit, push, and pull from your computer using GitHub Desktop
6. Overview of Git pull, commit, and push using the command line and Visual Studio Code (alternate methods to GitHub Desktop) (Evan)
    a. For more detail about Git, check the Get Git workshop videos.
    b. For more details about GitHub Pages, check the Go-Go GH-Pages workshop videos.

1:00 pm - 2:30pm - Objects and Metadata for CB-CSV

7. Discuss Objects
    a. What type of objects do you have?
    b. Where to host items
    c. Possibility of external items, curating from other collections
    d. Custom item types
    e. Processing using Rake tasks
8. Discuss CB-CSV Metadata concept and technical fields
    a. Google sheets tricks (Quick Tutorial/Presentation)
    b. Adding links for object_location, image_small, image_thumb in metadata CSV

2:45pm - 4:00pm - Collection Work Time - Basic Configuration

9. Initial collection set up (repeating steps from yesterday's CB-GH demo!)
    a. _config.yml
    b. Add metadata and objects
10. theme.yml
11. Config - csvs
    a. Customize nav
    b. Metadata config, markup mapping options and impact
    c. Explore how changes impact the default pages
    d. Think about how changes in the metadata could enable better use.
12. Other built in customization options
    a. Bootswatch (cool demo, but probably not for production!)
    b. Fonts

    c. Color theming
13. Push work to GitHub
14. Set up build using GitHub Actions
    a. "_config.yml" url and baseurl and noindex
    b. Option to Not build on GitHub at all! You can just keep working and previewing on your local machine.

Day 3: Wednesday, June 7 - Customizing and Creating Interpretive Content

Topics: Markdown, Liquid Includes, Jekyll Layouts, Bootstrap

Major Learning Objectives:

Conceptual

- *Understand the difference between CollectionBuilder and Jekyll*
- *Understand the nested ('russian doll') nature of a Jekyll project*
- *Be familiar with Markdown and how one can use it to interpret CollectionBuilder exhibits*

Technical

- *Be able to recognize Liquid within a Jekyll project*
- *Be able to locate an Include file within the repository*
- *Be able to find the data assets and manipulate them*
- *Be able to create a new page in the project*
- *Be able to reconfigure the home page layout using _layout/home_infographic.html*
- *Know how to write an about page and include an image from your collection.*

Day 3 Outline:

9:00am - 10:15am - Check in and introduction to technical components of CollectionBuilder

1. Morning check in
    a. Review process of editing CollectionBuilder repository on your computer
    b. Answer questions related to this process
    c. Demonstrate how to add new items and make changes to metadata in Google Sheets, and add the revised metadata CSV to your repository (Olivia)
    d. Demonstrate how to update Sheets from a local CSV
2. CollectionBuilder components and design overview - slides (Evan)
    a. Material of the web: HTML, CSS, JS
    b. Template components and recipes
    c. Where to find stuff in the template
    d. How to search around the template

e. Accessibility

10:30am - 12:00pm - Creating interpretive content + collection customization

3. Begin Customizing About page (Evan)
    a. Introduce Markdown (headers, paragraphs, links, lists) to write content
    b. Introduce Liquid Includes to add collection objects and bootstrap features
        i. Add an image (_includes/feature/item-figure.html)
        ii. Add a card (_includes/feature/card.html)
        iii. Update About nav (_includes/feature/nav-menu.html)
    c. Add a new page
    d. Update navigation with dropdown
4. Customize the Home page - bullets \ model (Devin)
    a. Jekyll Layouts (_layouts/home-infographic.html)
    b. Bootstrap columns and introduce Bootstrap classes
    c. Liquid Includes in layout
    d. Example customization: TimelineJS include on home page (_includes/feature/timelinejs.html)
5. How to create a custom Item page (i.e. object_template)
    a. Item layouts + item includes
6. Custom CSS
    a. Introduction to SASS / SCSS
    b. Using "_sass/_custom.scss"

1:00 pm - 2:30pm - Collection Work time - customize features and content

7. Work on your collections to add pages and interpretive content
    a. Customize your Home page layout by moving around/deleting/adding includes and changing the Bootstrap grid
    b. Add some content to your About page. At the very least we'd like you to create the following content in markdown.
        i. Write a couple paragraphs about your collection
        ii. Create a list or other markdown features
        iii. Add some includes
    c. Possible customization options:
        i. Repurpose the timeline
        ii. Create a new TimelineJS page
        iii. Create a new Cloud page
        iv. Add a dropdown to the Navigation Menu
        v. Add custom colors
        vi. Add custom bootstrap to layouts

2:45pm - 4:00pm - Discuss customization and deployment options

8. Discuss customization ideas
    a. What type of customizations would help communicate collection information and useability
    b. What type of interpretive content would be helpful
    c. What types of unique features would make it more interesting and engaging
9. Further details and options about how to build and deploy your site (Evan)
    a. Jekyll build (rake deploy) vs. jekyll serve
        i. Adding analytics, meta markup
    b. Deploying outside of GitHub Pages - objects and web site
        i. Limitations of GitHub hosting
        ii. Custom domains
    c. Using third party options (possibly with other types ...)

Day 4: Thursday, June 8 - Polish, Share, Discuss

Topics:

Major Learning Objectives:

Conceptual

- *Understand how Jekyll creates pages and site architecture using permalinks and layouts*
- *Understand how Jekyll uses the Liquid programming language to populate CollectionBuilder layouts*
- *Understand the difference between #collectionsasdata and collections in context*

Technical

- *Be able to use _data/theme.yml to customize the look of your project*
- *Be able to recognize Liquid within a Jekyll project*
- *Be able to locate an Include file within the repository*
- *Be able to find the data assets and manipulate them*
- *Be able to create a new page in the project*

Day 4 Outline:

9:00am - 10:15am - Check in and contexts for CollectionBuilder

1. Morning Check in
    a. Follow up on About pages and interpretive work
    b. Does everyone understand how the includes work? Where to find them?
2. Discussion of Collections as Data and collections in context
    a. Explore CB data features
3. Discussion of CollectionBuilder Types and use cases

a. GH
    i. Great for teaching, learning, prototyping, exhibits
    ii. Limitations = objects size, no image derivatives (i.e. thumbs)
b. Sheets! – new version currently in development, based on CB-GH, but can connect directly to a CSV hosted on the web (such as Google Sheet!).
    i. Quick demo (Evan)
    ii. Use for teaching - History 454 Mining the Archives
    iii. Use in other contexts (demo digital dramaturgy/incarcerated students project)
        1. Test Site for Incarcerated Students
            a. Site + spreadsheet
        2. Digital Dramaturgy spreadsheet + site
            a. Published CSV
c. CSV - different contexts
    i. More robust and flexible, supports image derivatives, relatively easy to customize item pages, can accommodate items from anywhere, larger collections, plugins support more features (such as generating pages from multiple CSVs)
    ii. Demonstrate moving files to server
d. Good place to ask questions: CB GitHub Discussions

10:30am - 12:00pm - Collection Work Time

4. Work on projects - options:
    a. Further customization
    b. Polishing
    c. Create another collection!
    d. Consult with CB team
    e. Review with partners

1:00 pm - 2:30pm - Show and tell

5. Show and Tell!
    a. Showcase the collection(s) you've created during this course
    b. Discuss further ideas, potential use cases, scholarship projects

2:45pm - 4:00pm - Finish show and tell + discuss static web use cases

6. Discuss static web landscape
    a. Lib-Static - highlighting lots of static web-based projects in DH, a viable and vibrant alternative.
    b. Learn-Static - Digital Humanities-focused static web teaching modules and project templates exploring the possibilities of static web in the classroom.

Day 5: Friday, June 9 - Wrap up

Topics: Share, debrief, reflect!

Day 5 Outline:

9:00am - 10:15am - Wrap up

1. Customizations and improvements for CollectionBuilder
    a. Road Map for the project
    b. What types of improvements would you all like to see?
    c. Ways to contribute to code, design, etc. going forward.
2. Assess the class: concerns, questions, or feedback?
3. Keep in touch! Check our CollectionBuilder contacts for details.

10:30am - 12:00pm - Institute Lecture (central)

1:00 pm - 2:30pm - Show & Tell Reception (central)

CollectionBuilder Resources

Websites and Repositories

- CollectionBuilder main site (info and documentation), https://collectionbuilder.github.io/
- CollectionBuilder GitHub organization (code repositories), https://github.com/collectionbuilder/
- CollectionBuilder presentations repository, https://osf.io/5sevc/
- Lib-Static website (community and resources around using static web methodologies for library and DH projects): https://lib-static.github.io/
- University of Idaho Digital Collections (mostly CollectionBuilder-based), https://www.lib.uidaho.edu/digital/collections.html
- Center for Digital Inquiry and Learning (University of Idaho digital scholarship unit, following mostly Lib-Static style development), https://cdil.lib.uidaho.edu/

CollectionBuilder Types

There are different versions or "types" of CollectionBuilder template depending on where you want to store the collection objects, your technical expertise, and the aims of your project. There are currently three main versions:

- CollectionBuilder-GH ("GitHub Pages") - a simplified template designed for free hosting on GitHub Pages that can be implemented without installing anything on your computer. It is intended as a pedagogical tool useful for classroom projects, teaching about digital libraries, or getting started with CollectionBuilder.
- CollectionBuilder-CSV ("CSV") - A robust, metadata-driven collection designed to help users build sustainable Stand Alone projects on a highly customizable base using the newest version of CollectionBuilder.
- CollectionBuilder-Sheets ("Sheets") - A super-simple version that lets you publish your Google Sheet (or any CSV) and add the link to the repository to generate a CollectionBuilder site. Best used for teaching and testing.
- CollectionBuilder-CONTENTdm ("Skin") - a template to create a new front end on top of existing CONTENTdm digital collections to provide a better user interface for browsing and discovery. It uses APIs to call images into the web pages from your existing repository.

Additionally, a complete digital library replacement with an ElasticSearch component providing central indexing is in a prototype stage under active development.

CollectionBuilder, Customized

"Storying Extinction: Responding to the Loss of North Idaho's Mountain Caribou" is a multidisciplinary digital humanities project that represents community response to the recent extirpation (2019) of southern mountain caribou from the South Selkirk mountains of North Idaho—the last caribou to inhabit the coterminous United States. The site centers a heavily customized CollectionBuilder item page on top of a leaflet map and allows users to move through the collection and map at the same time.

"Black History at the University of Idaho" is an ongoing digital exhibit that features archival items from the library and student-produced features out of U of I's Black History Research Lab related to Black students, faculty, and staff at the University of Idaho. This site uses CSV and features a number of the special features of a CB site, including a TimelineJS feature and several interpretive essays.

"Digital Library of Idaho" is a 'collection of collections' site that features digital collections from libraries and other memory organizations from across the state of Idaho. This site uses a customized version of CB-CSV, and its development was driven by collaborative metadata work almost entirely in Google Sheets.

"The Letters of Marie Mancini" is a collection of 17th-century letters written in French and Italian, and transcribed and translated into English. Item pages feature IIIF tiled images in an OpenSeadragon viewer alongside raw, edited, and transcribed letter text. Visualizations include a StoryMapJS instance that displays the locations in Europe where the letters were written. The site's browse page, item pages, maps, and glossary features are built with a customized CollectionBuilder codebase, and a custom Ruby script allows item data to be generated from letters' XML files instead of a CSV.

"Ella Fitzgerald Collection" is a small student-created collection containing images of Fitzgerald's clothing, album liners, and historic photographs held at the University of Idaho Special Collections and Archives. This site uses a customized version of CB-CSV to produce compound item pages which allow for the inclusion of multiple images that show different views of a single clothing item, while the essay on the "About" page is a good example of how a student might write *with* a collection by including collection items and quotes from Fitzgerald to tell a brief history of the artist's career.

“Civilian Conservation Corps in Idaho” is a project in collaboration with a U of I faculty member who wished to share materials accumulated during archival research. This enables an opportunity to go beyond the final publication, providing access to a unique curated collection.

“1918 Flu Pandemic Collection” is a special exhibit created by Special Collections staff to highlight timely, thematic content. The agility and simplicity of CollectionBuilder allows staff to learn how to contribute ideas, content, and long form writing to efficiently collaborate with digital librarians to

bring this type of collection to life. We would not have been able to publish this sort of collection using our traditional repository workflow.

"Historic Japanese Ceramic Comparative Collection" is an early customized version of a CollectionBuilder-SA template developed in collaboration with a graduate student in Archeology to publish her unique research beyond a dissertation. This highlights the project's flexibility to adapt to different content types and means of navigation. The graduate student was able to meaningfully contribute to the collection site by creating data and content, without needing skills in web development.

"University of Idaho: Then and Now" is a project created by an undergraduate student during a fellowship with the Center for Digital Inquiry and Learning. The student explored archival images of campus, then rephotographed the locations to provide "then & now" comparisons. This shows how the CollectionBuilder template can be quickly modified and extended with new features, such as KnightLab's JuxtaposeJS.

"Mining History in Idaho" is a student-led collection created by an undergraduate history course using an early version of CollectionBuilder-GH. Creating the collection provided students an experience in archival research, digitization, metadata creation, and web publishing.

"Adult Salmon and Steelhead Migration Studies: 1996-2014" is an early stand alone collection built from metadata and PDFs submitted by an ecology lab to preserve research for long term open access. This highlights the possibilities to provide flexible institutional repository services that represent the unique contexts of research that might otherwise never be published online.

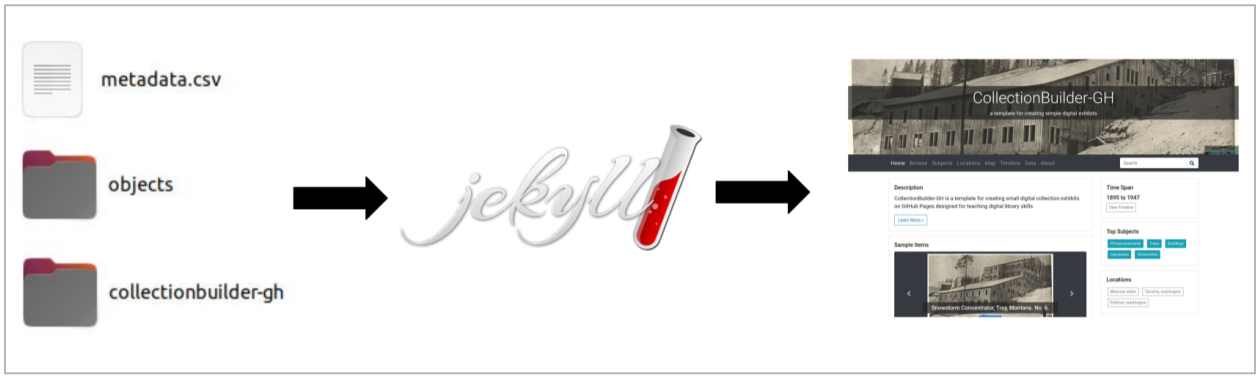
CollectionBuilder Basics

CollectionBuilder is a set of flexible open source templates for creating digital collection and exhibit websites that are driven by metadata and powered by modern static web technology. To generate a digital collection website, users:

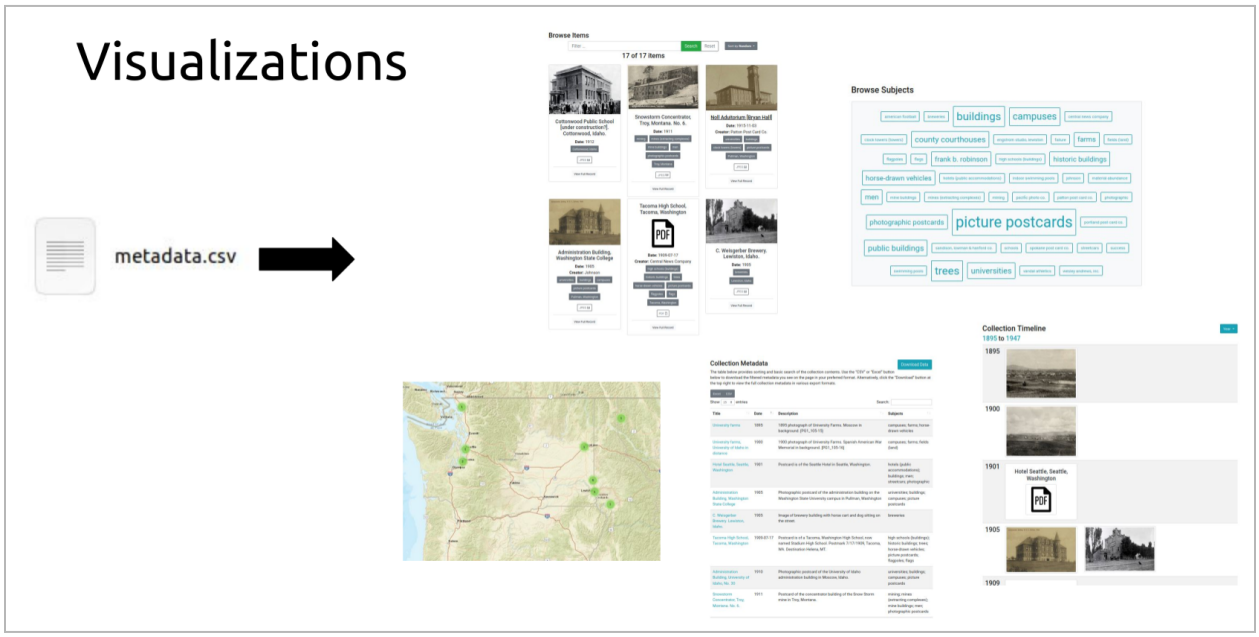
- Create metadata in a spreadsheet
- Organize a corresponding folder of digital objects (images, PDFs, videos, etc)
- Make a copy of the CollectionBuilder template code
- Configure and customize the site using built-in options
- Write contextual content in Markdown

Once set up, the CollectionBuilder project is transformed by the popular static site generator Jekyll (installed on a laptop or used through an automatic build process such as GitHub Pages hosting service) into a complete website for browsing and contextualizing the collection. The output is static HTML that can be copied to any basic web server, or hosted for free on GitHub Pages, providing a

simple, preformant, and secure website—alongside clean data and metadata ready for long term digital preservation.



CollectionBuilder templates are a packaged folder of plain text files, including modular chunks of HTML, CSS, and JS, and helpful development libraries such as Bootstrap. Users need not know anything about Jekyll, Liquid, or the other tools that power CollectionBuilder to get started. Instead, they begin by focusing on their collection's metadata and digital objects independent of the system, then follow step-by-step documentation to add them to the project template.
The CollectionBuilder code consumes the collection metadata added by the user to automatically generate browsing features, items pages, data derivatives, and rich SEO markup. Metadata drives the core of the user interface, creating interactive browsing pathways and visualizations that encourage visitors to explore and discover content while also representing overall context, transforming high quality description into a rewarding user experience.



CollectionBuilder Workflow

To demonstrate how CollectionBuilder works, this section walks through our current pedagogical version, CollectionBuilder-GH, a tool intended as a simple template for hands-on teaching about digital libraries.

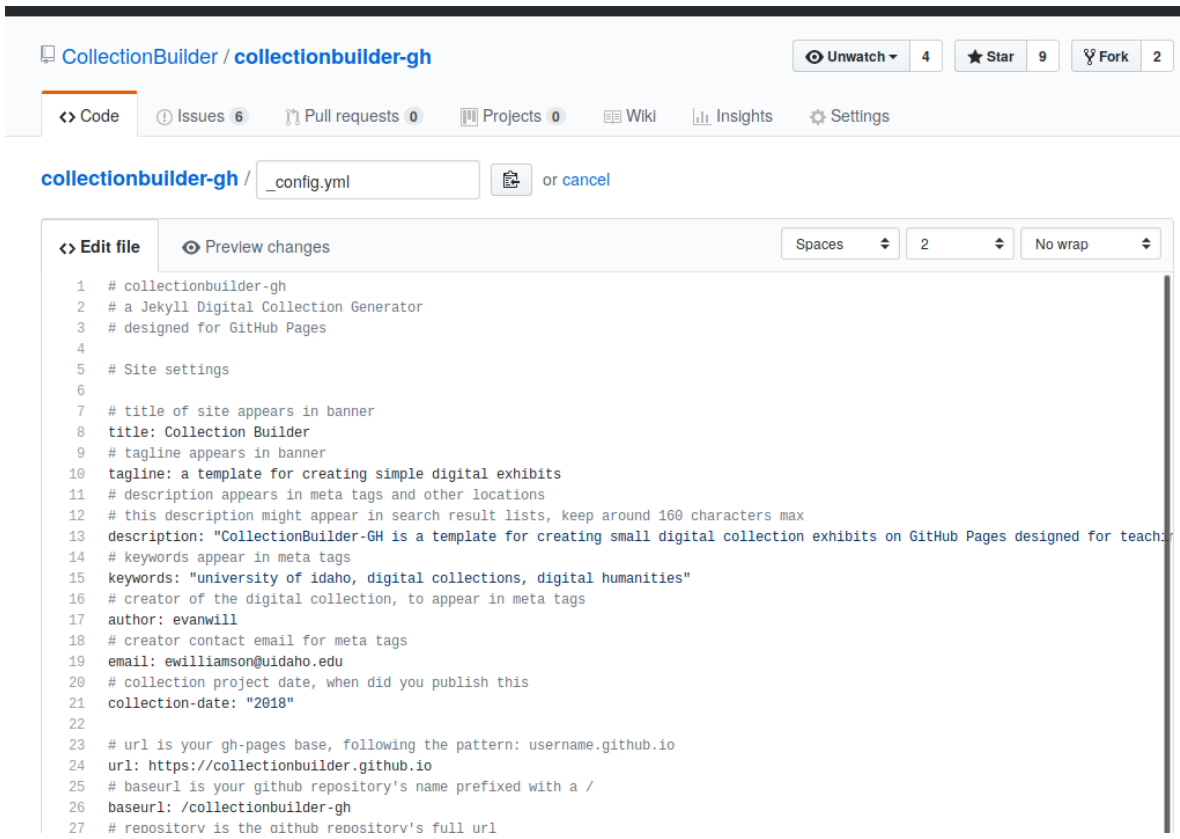
CollectionBuilder-GH can be used in a workshop or classroom setting to take participants through digitization and metadata creation, to having a live digital collection website hosted on GitHub Pages without installing any software (this contrasts with the other CollectionBuilder versions which rely on Jekyll being installed on a local development machine). The only requirement for both instructor and participants is a GitHub account and a web browser. Similar learning experiences often use Omeka or other DAMS/CMS platforms with extensive infrastructure requirements that are overkill for one-off projects. Although these platforms feature familiar GUI administration interfaces, they are not necessarily simple to learn and customize. CollectionBuilder-GH aims to be well documented and easy to configure by following the example—if you match the metadata template, a fully functional website will be automatically generated. Customization is learned in additional small steps, encouraging scaffolded learning about web and data fundamentals. A project in "minimal computing", CollectionBuilder-GH provides a depth of learning opportunities, allowing users to take complete ownership over the project while making their work open to the world.

1. Setup: After getting set up with GitHub accounts and orientation, users start their new project by creating a copy of the code on GitHub by clicking the "Use this Template" button, https://github.com/CollectionBuilder/collectionbuilder-gh

2. Prepare Objects: Users then prepare their folder of digitized objects (generally images and/or PDFs) following the documented standards, and upload them into their project repository's "objects" directory. This is an opportunity to teach data skills such as file naming, preservation formats, and media editing.

3. Prepare Metadata: Users prepare a spreadsheet of metadata for the objects, following the CollectionBuilder metadata template (which is based on digital libraries best practices and standards). We typically use Google Sheets, allowing easy collaboration with groups of participants. This hands-on digitization and metadata creation experience helps reveal the real labor and decision-making processes that go into the creation of digital library data. Skills learned manipulating media files and working with spreadsheets are more transferable data fundamentals than comparable workflows in CMS interfaces which focus on forms and clicks. Once the metadata is prepared, it is downloaded/exported as a CSV, and uploaded into their project repository's "_data" directory.

4. Configure Site: With the well-structured data prepared, users can begin working on the website configuration, including the "_config.yml" which provides the base settings for Jekyll websites, such as site "title," "tagline," and "description." Config files can be edited using GitHub's web interface.



Editing the _config.yml using GitHub's web editor

5. Add Descriptive Content: Users can edit content pages, such as the About page, to provide context for their collection. Page content is styled using Markdown, providing an easy to learn and write markup language.

6. Activate GitHub Pages: With the click of a button, users can activate GitHub's free hosting service and have a live website in seconds. In the background, the platform automatically runs the static site generator (Jekyll) over the source code, outputs a complete website (HTML, CSS, JSON, and JS files), and serves it up to the world. Users navigate to their site to discover their new digital collection and explore the visualizations. They will likely discover interesting metadata anomalies that were not apparent when working on the spreadsheet—a teachable moment about how to debug your project!

By creating a collection using CollectionBuilder, students develop interwoven technical and critical skills, including fundamental data literacies related to controlled vocabularies, unique identifiers, and descriptive practice. These lessons are reinforced when their metadata is transformed into a digital collection on the web, inevitably surfacing anomalies, breakages, and misrepresentations tied to issues in the metadata that they return to the spreadsheet to fix. Small "next steps" invite students to start

learning more about the templates generating the website, encouraging incremental development of further web skills.
Outside of the classroom or workshop setting, we still see CollectionBuilder as a tool rooted in learning. CollectionBuilder is unlike common Content Management Systems (CMS) or Digital Asset Management (DAM) platforms that most institutions use—it is not a visual GUI tool and there is no admin interface, which can intimidate and contribute to an initial learning curve. However, it is designed to be 'do-able' and accessible for librarians and digital humanities practitioners—and once understood, users will be rewarded with a flexible and sustainable template for creating digital collections, as well as data and web skills transferable to any digital project. This opportunity for professional development provides unique agency for librarians to take full control over their systems and pursue new initiatives and ideas equipped with CollectionBuilder components as a recipe book of solutions.

Using other versions of CollectionBuilder follows a similar workflow as described above, but allows for further customization, features, and optimization of the site. Users edit the template code on their own computers and run Jekyll to provide a local development server to test their work. The code aims to be modular, understandable, and well-documented with the goal to be sustainable for a low-resourced digital libraries team. Once a collection is ready to deploy, the developer uses Jekyll to build the self-contained static website and copies the files over to a basic web server or hosting service.

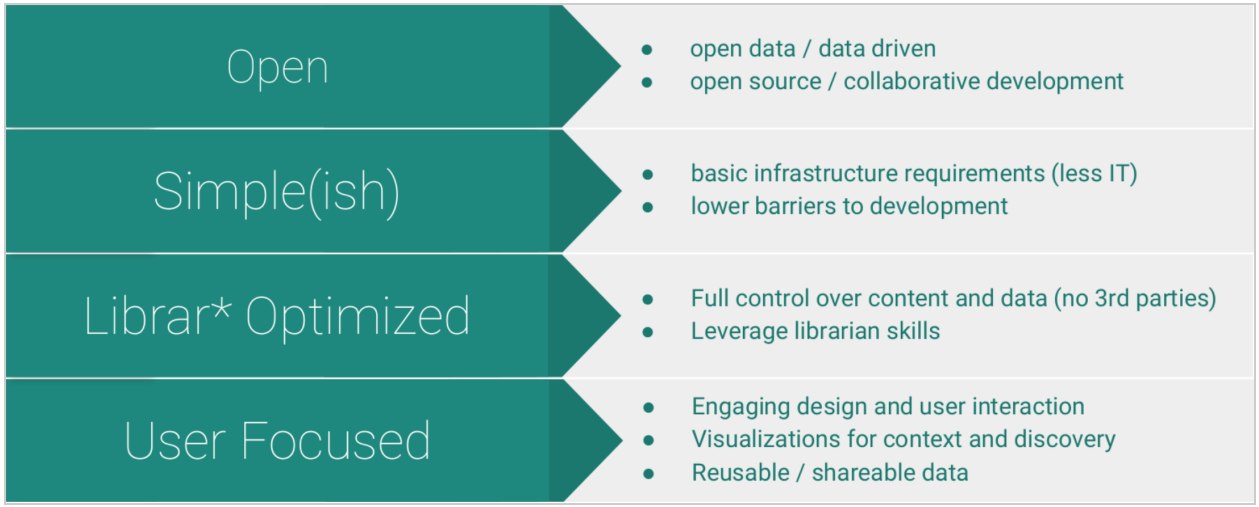
The Lib-Static Methodology

Check out our new site! https://lib-static.github.io/

Since around 2015 static site generators and the "JAMstack" approach have exploded in the web development landscape—utilizing simplified markup, plain text data files, and web APIs to create complete websites without the need for complex server applications, databases, or content management systems. Rather than relying on server-side processing to generate a dynamic page on the fly for each user request, static web tools "pre-render" every page into HTML, CSS, and JS files that can be delivered directly to users with speed, efficiency, and security from the most basic web servers or services. This modern static web approach provides high performance for users, minimal infrastructure requirements for IT, and lower barriers for developers.

Eager to explore this potential in the library context, faculty librarians at the University of Idaho (U of I) have been developing digital collections, digital humanities projects, and instructional content using static web tools for more than five years. Informed by the philosophy of minimal computing, they have been documenting the "Lib-Static" methodology to begin building a community of practice centering digital infrastructure approaches around the unique needs, values, and opportunities of libraries.

The primary principles of the methodology are summarized in the visualization below:



Simply stated, Lib-Static-inspired tools seek to apply the techniques of the modern static web approach to pragmatically solve problems in the digital library ecosystem. It differs greatly from the predominant model for platform and tool building for academic libraries as it does not require complex infrastructure nor extensive IT staffing (or third party vendors) to implement and maintain the systems put into use. Instead, the Lib-Static approach focuses on practical, sustainable workflows using data-driven static web templates hosted on simplified infrastructure while leveraging the in-house specialized skills of librarians in metadata, data, and organization. This provides librarians unique agency and ownership over their systems, as well as meaningful opportunities for professional development leading to fundamental digital skills (instead of the "buttonology" of a single platform). The focus on clean data and simple systems enables a more agile and responsive approach, allowing the iterative development of features, gradual acquisition of developer skills, and flexible migration between hosts without the need for deep investment.

Lib-Static acknowledges that all digital projects require investment in learning and seeks to maximize the local impact and value of learning during the process, while establishing technical solutions and social workflows that more closely match the structure of academic work cycles and needs. The simplified infrastructure and development environment of static web approaches are uniquely suited to enable:

- Periods of focused development and collaboration, followed by much longer periods of minimal maintenance.
- Project work focused on creating data independent of platform, which simplifies the initial infrastructure decision points and overhead while ensuring preservation- and migration-ready content.
- Rapid design and data iterations—building projects in smaller steps allows data to be published faster, getting feedback earlier with less initial investment yet future opportunity for phases of progressive enhancement.

- A focus on modular components, templates, and recipes that encourage learning investment on one project leading to efficiencies on another, building complementary work that powers further research and learning.
- Public documentation and sharing, making investment more reusable while creating artifacts of learning alongside research and publications.

The Lib-Static methodology has grown out of U of I librarians' experience collaborating with faculty, students, staff, and other organizations to create digital collection, digital scholarship, and teaching projects. Bogged down in the process of standing up and maintaining heavy infrastructure platforms for one off projects, they found a real need for a more flexible and sustainable approach to creating websites. The static web approach has freed up energy and time to pursue new initiatives and ideas, leading to unique collaborations and learning opportunities. Projects have included an open music text book, workshop outlines, interpretive oral history projects, a scholarly translation edition, scientific research archives, classroom digitization experiences, community digital collections, Special Collections exhibits, and the main library website--the Lib-Static methodology has truly infused our work! Each collaboration has advanced new solutions, features, and concepts which have in turn contributed to the development of several refined templates designed as reusable tools or bases for adaption, including Oral History as Data and CollectionBuilder.

Readings

Becker, Devin, Evan Williamson, and Olivia Wikle. "CollectionBuilder-CONTENTdm: Developing a Static Web 'Skin' for CONTENTdm-based Digital Collections," *Code4Lib Journal* 49, 2020, https://journal.code4lib.org/articles/15326

Dombrowski, Quinn. "Sorry for all the Drupal: Reflections on the 3rd anniversary of 'Drupal for Humanists,'" *Quinn Dombrowski* (blog), November 8, 2019, http://www.quinndombrowski.com/?q=blog/2019/11/08/sorry-all-drupal-reflections-3rd-anniversary-drupal-humanists

Gil, Alex. "The User, the Learner, and the Machines We Make." *Minimal Computing*, May 21, 2015, https://go-dh.github.io/mincomp/thoughts/2015/05/21/user-vs-learner/

Nowviskie, Bethany. "speculative collections." *Bethany Nowviskie* (blog), October 27, 2016, http://nowviskie.org/2016/speculative-collections/.

Russell, John E., and Merinda Kaye Hensley. "Beyond buttonology," *College & Research Libraries News*, 2017, https://crln.acrl.org/index.php/crlnews/article/view/16833/18427

Varner, Stewart. "Minimal Computing in Libraries: Introduction." *Minimal Computing*, January 25, 2017, https://go-dh.github.io/mincomp/thoughts/2017/01/15/mincomp-libraries-intro/

Visconti, Amanda. "Introducing Static Sites for Digital Humanities Projects (why & what are Jekyll, GitHub, etc.?)," *Literature Geek*, December 8, 2015, http://literaturegeek.com/2015/12/08/WhyJekyllGitHub

Wikle, Olivia, Evan Williamson, and Devin Becker. "What is Static Web and What's it Doing in the Digital Humanities Classroom?" *dh+lib*, Special Issue: Literacies in a Digital Humanities Context, 2020, https://dhandlib.org/2020/06/22/what-is-static-web-and-whats-it-doing-in-the-digital-humanities-classroom/

Williamson, Evan, Olivia Wikle, Devin Becker, Marco Seiferle-Valencia, Jylisa Doney, and Jessica Martinez, "Using Static Web Technologies and Git-based Workflows to Re-Design and Maintain a Library Website (Quickly) with Non-Technical Staff." College & Undergraduate Libraries, 2021, DOI: 10.1080/10691316.2021.1887036 (post print: http://doi.org/10.17613/v30m-mx03)

code{4}lib JOURNAL

Issue 49, 2020-08-10

CollectionBuilder-CONTENTdm: Developing a Static Web 'Skin' for CONTENTdm-based Digital Collections

Unsatisfied with customization options for CONTENTdm, librarians at University of Idaho Library have been using a modern static web approach to creating digital exhibit websites that sit in front of the digital repository. This "skin" is designed to provide users with new pathways to discover and explore collection content and context. This article describes the concepts behind the approach and how it has developed into an open source, data-driven tool called CollectionBuilider-CONTENTdm. The authors outline the design decisions and principles guiding the development of CollectionBuilder, and detail how a version is used at the University of Idaho Library to collaboratively build digital collections and digital scholarship projects.

by Devin Becker, Evan Williamson, and Olivia Wikle

Context

Unsatisfied with the limited options for customizing collections hosted in their digital asset management (DAM) system, CONTENTdm, librarians at the University of Idaho developed CollectionBuilder-CONTENTdm, an open source approach that uses CONTENTdm's API to build customized, discovery-focused web pages and visualizations on top of a collection's metadata. Acting as skins, these overlaying web pages provide an alternative interface for users to explore collection content and context, without directly interacting with CONTENTdm's website.[1] The digital collection websites built with this tool are generated from CSVs using a static web generator, and provide users with total access to and control over all code and data. This article briefly details the background of the "skin" approach, provides an in-depth look at the code making up the CollectionBuilder-CONTENTdm template, and lays out the workflows used at the University of Idaho to build and maintain the University of Idaho Digital Collections.

Why a Skin?

While CONTENTdm facilitates certain features that are difficult to replicate via other tools or services—namely full-text and cross-collection searching, media object storage and retrieval, and established workflows for ingesting/exporting/exposing metadata and objects—we developed skins for our collections for three main reasons:

1. We don't believe the default CONTENTdm interface is particularly user-friendly, and we don't find the customization options effective or usable enough to accommodate the discovery-focused designs we aim for with our digital collections.
2. We believe our collections (and archival collections in general) are truly 'special' and as such demand and reward customized treatment to relate their extraordinary nature to our users. Items in our collections are not disembodied database objects, and we want to ensure context is communicated to users.
3. The SEO practices and machine-readable markup, or lack thereof, of CONTENTdm do not promote the discovery of individual collection items via external search engines.

The 'skin' approach allows us to utilize the search and storage features of CONTENTdm that we find difficult to reproduce, while allowing us to curate a user experience for the collection that matches the quality and interest of the content we are charged with stewarding.

Historical Development

We began using this skinned approach with our digital collections in earnest around 2012. Starting by hand-coding websites using basic design features, we quickly progressed to exporting metadata from CONTENTdm as XML and using XSLT worksheets to generate the overlaying HTML pages for each customized collection. Some of these XML/XSLT techniques were described in a 2014 Code4Lib article detailing the development of the Latah County Oral History Collection.[2] Template elements, such as headers, footers, and analytics, were added using PHP includes to simplify maintaining consistency. Links to individual items pointed to the CONTENTdm item page. This workflow was effective in many ways but required the maintenance of many different XSLT files that were then run against metadata XML files. Since each transformation had to be run individually, and the outputs had to be moved to a development server to test the code, viewing results from iterative development was slow. The system was also highly idiosyncratic, making it difficult to collaborate with other developers.

In 2015, we started to experiment with employing modern static web generators to build our collections and found that this method of development resolved the issues of version control and collaboration that hindered our XML/XSLT workflow, allowing us to consolidate our code and create collections that are at once more uniform in terms of layout and individualized in the way of content. The concepts behind our original skins were not abandoned: the visualizations, designs, and patterns of use have inspired and informed many of the components we have developed for CollectionBuilder.

We detail these components below, including brief discussions of how CollectionBuilder's current code has been influenced by past tools and methods as our new approach has matured into a project capable of efficiently generating our new collection skins. If you'd like to fully understand how the system works, and its various requirements, please peruse the documentation available on the CollectionBuilder website.

Code

CollectionBuilder-CONTENTdm uses the static web generator Jekyll together with the templating language Liquid to build the layouts and features of the generated websites. Three main components are provided or edited by a user to build an individual digital collection website:

- a CSV file containing the collection's metadata;
- two main configuration files (_config.yml and _data/theme.yml);
- and a series of page-specific config files (in CSV format) that determine what content appears on certain pages.

During generation, Jekyll exposes the metadata CSV from the project's "_data" directory, making it available for use in pre-built templates throughout the project to build out visualizations. Despite much of the code being pre-built, each page can also be flexibly modified using the user-generated configuration files. This enables the tool to accommodate a wide variety of collection types and content without a user needing to edit the base code. It also allows us to develop the entirety of our digital collections site via one Git repository with a branch for each collection.

Collection as Data (and more data and more data …)

CollectionBuilder's visualizations, features, and item pages demonstrate an expression of the "collections as data" ideal in that they are fundamentally metadata- and data-driven. Using Liquid templates, the code for each visualization creates specialized derivatives of metadata that are consumed to generate the web page. This approach puts the focus on metadata as data, rather than loosely structured description, and rewards high quality metadata by creating intricate browsing pathways for users.

Jekyll and Liquid are particularly suited to this sort of data transformation, able to generate new formats that can be reused in other applications. Rather than hide this data, we made the decision to expose it, making a variety of publicly re-usable derivatives easily downloadable from the site. For example, below is a list of the outputs (with links) that are listed at the bottom of the home page for the Archival Idaho Collection:

- Metadata CSV – all metadata fields in CSV format, configurable by the "metadata-export-fields" variable in theme.yml
- Metadata JSON – all metadata fields in JavaScript Object Notation (JSON) format
- Subjects JSON – unique subject terms and their counts in JSON
- Subjects CSV – unique subject terms and their counts in CSV
- Geodata JSON – a GeoJSON file with geographic coordinates and associated item metadata
- Locations CSV – unique location terms and their counts in CSV
- Locations JSON – unique location terms and their counts in JSON
- Timeline JSON – a JSON file formatted for use with TimelineJS
- Facets JSON – unique terms and counts for all metadata fields listed in the "metadata-facets-fields" variable in theme.yml

These metadata downloads and a table representation are further highlighted on the Data page. The web table is powered by DataTables with the JS code optimized to handle thousands of items; it allows users to filter and export subsets of the metadata records. Of course, all of these data formats are not necessary for every collection, so the tool will only build each data file if the related web page is also included in the navigation configuration file (_data/config-nav.csv), which controls which pages display as headers for the site. This means, for instance, that if one were to remove the map page from their navigation configuration file, the system would not expose a GeoJSON file for the collection.

In designing the tool, we took the "collections as data" mantra to heart. We hope this explicit acknowledgment of the data underlying the digital collection will encourage others to creatively reuse and rethink our repository. We also see this feature as an acknowledgment of the reality of migration: no platform is future proof–instead keeping the focus on investing in quality machine- and human-readable data in a variety of formats from the start ensures the collections we steward are ready for migration, preservation, and reproducibility.[3]

Home Page, a Bento Box of Options

First impressions, as we all know, are important. Most digital collections, however, are introduced poorly or not at all. Out-of-the-box CONTENTdm collections, for instance, feature a "landing page" that provides a space for a general textual description of the collection, that with some work can be customized with a few visual features (e.g. 2016 example of our Higgins collection). Alternatively, many more complex systems simply drop a user into a catalog-like presentation of the collection, eschewing contextual information altogether. We believe our collections—and, for that matter, the collections held by libraries and other institutions like them across the US—deserve better introductions than they currently receive. The design of our various skins throughout the years reflect that belief, as does our current iteration.

Our early skins often included a feature such as a carousel[4], map[5], timeline, or interactive display[6] to draw users into the collection. CollectionBuilder sites go a bit further, providing both an eye-catching featured image and contextual information about the collection. The tool uses a collection's metadata to generate an overview of the collection's era, location, subject matter, size, and content, and provides entry links from the overview information to encourage exploration. In some ways this is influenced by the popular concept of "dashboards," but rather than forefronting administrative and use data, we are summarizing metadata to create a simple visualization of overall context for the collection.

Figure 1. The CollectionBuilder-generated Home page for our Barnard-Stockbridge Photograph Collection.

The home page is customizable via the configuration of the theme file (_data/theme.yml). The theme allows a user to choose the featured image, determine the content of the carousel, and adjust the number and/or content of the featured subjects and locations. This allows for a great deal of customization and impact without modifying the base code. Choosing a featured image, for instance, is an important act of curation for the collection, as derivatives are used to represent the collection in meta tag markup (which will be displayed on social media). The theme also allows one to adjust the size and placement of that image, which has a significant impact on the look and feel of the home page.

Below the featured image, the infographic layout is made up of a series of modular cards presented in a bento-box style populated by automatically generated content. For example, "carousel" provides a Bootstrap carousel with a randomly selected group of images with direct links to the items. The 'Time Span' card calculates the date range of the collection and links to the Timeline visualization page. The code in our "Top Subjects," "Top Locations," and "Objects" cards calculates the unique values in specific metadata fields and provides links to the Browse page which will sort to that grouping. These cards can be swapped out depending on which work best with the collection's metadata by replacing Jekyll _include commands — {% include index/[filename].html %} — on the home-infographic-layout.html file, which is contained in the _layouts folder. This enables the page to be quickly rearranged using Liquid includes and Bootstrap columns to provide variety.

Browse Page as Discovery Engine

Early versions of the skin often featured a "browse all" page with all collection images represented on an (almost) infinitely scrolling page. For some collections, all items were further represented as cards with image and basic metadata displayed. These cards were created using XSLT from the XML metadata, and arranged on the page using the jQuery plugin Isotope with masonry layout. Isotope allowed for randomizing the items and filtering using a simple search box. These early pages allowed users to seamlessly browse through all items at a larger scale as an alternative to clicking through page after page of thumbnails in the CONTENTdm interface "browse."

Building on these earlier designs, we developed a Browse page that provides cards for every item in the collection. The metadata featured on each card is fully configurable (via the config-browse.csv in the _data folder), and field information can appear as textual or be represented as a button that, when clicked, instantly filters the page (see Figure 2 for an example of the result). This filtering feature is central to CollectionBuilder sites, as links on almost every page lead back to filtered versions of the Browse page. This allows users to quickly explore groupings and subjects that they discover while browsing items across the site. The page tracks filtering via URL hashes, making any filter shareable by copying the link, and allowing other pages to link into the filtered view.

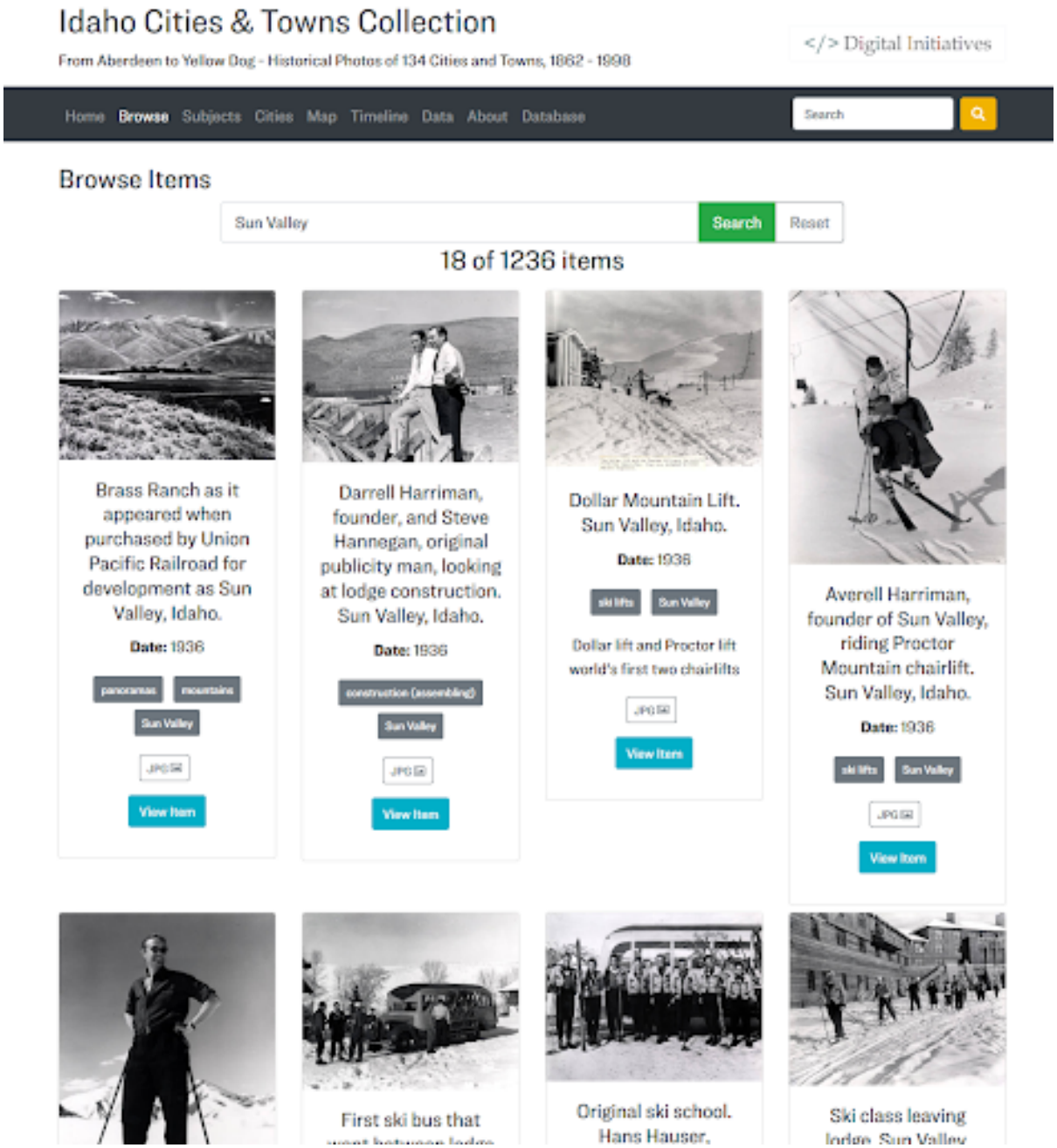


Figure 2. A portion of the Browse page for the Idaho Cities & Towns Collection, filtered to those items located in Sun Valley.

To make this visualization scaleable to thousands of objects[7], the creation of item cards and filtering is handled by JavaScript. Following options set in the config-browse.csv file, Liquid is used to create a subset of metadata in the browse-js include as a JS variable that then drives the visualization. The config file also controls the metadata fields which are displayed on the item card, using Liquid to manipulate the JavaScript function that creates each card. This function is kept fairly simplistic to enable easy modification of the card contents in cases where the automatic configuration is not flexible enough for the needs of the collection. The aim here, and throughout the codebase, is to keep the JavaScript simple and well commented enough that a typical digital librarian can figure it out, even if they are not expert developers.

Word Cloud (Still a really good visualization)

Word clouds are a common, yet useful way to quickly visualize word frequency in text. Applied to a metadata field, they provide a simple representation of the unique facets of a collection, allowing users to get an overall sense of the content, while also surfacing the unexpected. Some of our early skin sites featured word clouds of subject terms inspired by the Tagcrowd.com implementation.[8] Each subject term in those early word clouds linked to a CONTENTdm search, thus sending users out of the skin to the database view. Building on this concept, CollectionBuilder contains a flexible cloud layout, designed as a template that can generate a word cloud from any relevant metadata field, such as subjects, locations, or creators. Each unique term is rendered in the cloud as a button (highlighting its "clickable-ness"), with font size scaled to its relative frequency, and given a hyperlink to the Browse page which will display the related group of objects.

Despite the final result being a fairly simple visualization, this is one of the more complex pages to build, given the need to calculate unique terms and counts from the metadata fields which may contain thousands of individual values. Our first implementation was done entirely in Liquid, requiring several "for" loops within loops and a hacky data structure (as Liquid only supports a basic form of an array). A version of this Liquid routine is used in the lightweight version of CollectionBuilder, CollectionBuilder-GH, which is designed to run on GitHub Pages where Jekyll plugins are not allowed. We include it here, as it demonstrates a complex use of Liquid:

```
---

# find and count unique subjects used in the metadata

---

{%- if site.data.theme.subjects-page == true -%}

{%- assign cloud-fields = site.data.theme.subjects-fields | split: ";" -%}

{% comment %} Capture all cloud terms {% endcomment %}

{%- assign raw-terms = "" -%}

{%- for c in cloud-fields -%}

{% assign new = site.data[site.metadata] | map: c | compact | join: ";" %}

{% assign raw-terms = raw-terms | append: ";" | append: new %}

{%- endfor -%}

{%- assign raw-terms = raw-terms | downcase | split: ";" -%}

{% comment %} Clean up raw terms {% endcomment %}

{%- capture terms -%}{%- for t in raw-terms -%}{%- if t != "" and t != " " -%}{{ t | strip }};{%- endif -%}{%-

{%- assign terms = terms | split: ";" | sort -%}

{%- assign uniqueTerms = terms | uniq | sort -%}

{ "subjects": [

{% for u in uniqueTerms %}{% assign count = terms | where_exp: 'item', 'item == u' | size %}{ "subject": {{ u

{% endunless %}{% endfor %}

]

}{%- endif -%}
```

Liquid iteration tags like those used above, however, are fairly slow, which can lead to unreasonably long build times as the size of projects or data increase. In our initial iterations this slowness was compounded, as code throughout the repository calculated unique terms for several data outputs in addition to the Cloud page. As we began redesigning larger digital collections, the speed of Liquid became a limitation, making iterative development cumbersome as build times soared. At first, we would use smaller subsets of the metadata and turn off calculating clouds during development so that rapid iteration was still possible. One of the advantages of working with CollectionBuilder, however, is the way it exposes issues (and interest) in the metadata, so working with subsets is a suboptimal solution. This led us to explore methods to optimize the Liquid code, such as replacing "for" and "if" statements with equivalents using "where" or "where_exp," which are significantly faster and minimize iterations.

Even with optimization, using Liquid for calculations within Jekyll slows build time significantly—a much more efficient solution is to use a Jekyll plugin. Jekyll's Ruby-based plugin system provides a means to add custom functionality injected into the generator engine at build time. A large ecosystem of formally packaged plugins exist, or any new plugin can be written in Ruby and added to the project "_plugins" directory. Any calculations completed directly in Ruby will be exponentially faster than complicated Liquid routines.

After much experimentation, we developed a plugin "array_count_uniq" that adds a new Liquid Filter to the project environment. To use it, we first gather the desired data as a Liquid array, then apply the new filter:

```
{{ myArray | array_count_uniq }}
```

The filter will return a hash of the unique values and their frequency counts which can then be iterated over like any other array to use the calculated values in the page template. Following CONTENTdm conventions, we use semicolons to denote multivalued fields. For example, an average value in the Subject column might look like "Idaho; Potatoes; Mountains," which is obviously three separate subject terms, not a single value. Thus, to prepare the array, the cloud layout uses "map" to extract values from the desired field(s), joins them with a semicolon, then splits on semicolon to create a unified array of all terms which can then be passed to the "array_count_uniq" filter. Using the filter on a single metadata field ("example_field") would look like:

```
{% assign uniqueHash = site.data.metadata | map: "example_field" | join: ";" | downcase | split: ";" | array_co
```

The implementation in the cloud-js include is more complicated because we support combining multiple metadata fields into a single visualization.

This plugin filter reduces build times exponentially, completely removing the old bottleneck of calculating unique counts. Since the plugin is fairly simple and idiosyncratic to the needs of the CollectionBuilder project, we haven't independently packaged it—we consider it part of the template and it can be found in the _plugins folder. We see the creation of additional plugins as a way forward for making CollectionBuilder more efficient, but hope to balance it against our aim to keep the overall project complexity low.

Seeing Space: From Fusion Tables to Leaflet

When our early PHP-based skins were developed, creating your own interactive map layer was a very challenging undertaking. So when Google Fusion Tables launched in 2009,[9] the web service was embraced by librarians looking to build map visualizations using only a spreadsheet and bit of configuration on the platform. We implemented Fusion Tables extensively for our digital collections, customizing Google API JavaScript examples to load Fusion Tables-based maps on our collection pages. Like so many other Google products, however, Fusion Tables was terminated in December of 2019, and all our old map features are now obsolete.

Luckily, already growing cautious of 3rd party services, we had begun moving our map applications to self-contained JavaScript using the open source library Leaflet.js in 2017. Leaflet is efficient, well documented, and has a robust plugin ecosystem, making it relatively easy to implement custom map features on static pages using openly available tile layers.

To provide data to the map, we use a Liquid template to generate GeoJSON features for the items that contain latitude and longitude metadata values. Other descriptive metadata configured in "config-map.csv" is included on popups from the map pins or used in searching. A GeoJSON file with more complete metadata is also generated in the data exports, which can be easily used in other projects and analysis. Using Liquid templates with the metadata is a practical way to carry out these types of data transformations, as can be seen in the code to generate GeoJSON:

```
---

# generate geojson data for collection items with lat-longs

---

{%- assign items = site.data[site.metadata] | where_exp: 'item','item.latitude != nil and item.longitude != ni

{%- assign fields = site.data.theme.metadata-export-fields | split: "," -%}

{

"type": "FeatureCollection",

"features": [

{% for item in items %}

{

"type":"Feature",

"geometry":{

"type":"Point",

"coordinates":[{{ item.longitude }},{{ item.latitude }}]

},

"properties":{

{% for f in fields %}{% if item[f] %}{{ f | jsonify }}: {{ item[f] | jsonify }},{% endif %}

{% endfor %}

"reference_url": {{ '/items/' | absolute_url | append: item.objectid | append: '.html' | jsonify }}

}

}{% unless forloop.last %}, {% endunless %}{% endfor %}

]

}
```

Three Leaflet plugins add further functionality to the map visualization: 1) leaflet-fusesearch which provides basic search functionality for the items displayed on the map; 2) Leaflet.markercluster, which clusters individual items on the map, cleaning up the display and significantly improving performance and useability for large collections; 3) Leaflet.MarkerCluster.Freezable, which allows clustering to work alongside search by temporarily unclustering to display query results. To make working with Leaflet easier, we expose the basic map configurations and plugin options as values in the theme.yml file, allowing you to quickly test different settings for each collection. To ensure missing configuration options won't break the visualization, we set sane defaults using Liquid's "default" filter.

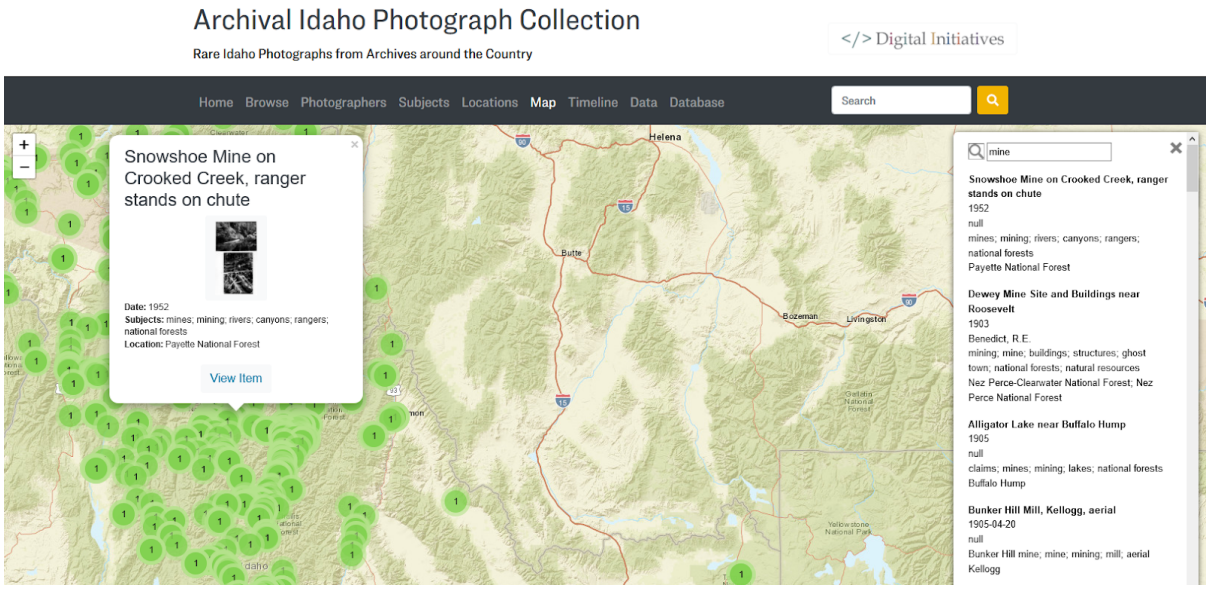


Figure 3. An example CollectionBuilder Map page from our Archival Idaho Photograph Collection. Leaflet plugins allow clustered map items to be searched.

Traveling Time: From Simile Timeline to Simple Table (+ timelinejs)

It seems dated now, but Simile's JavaScript Timeline visualization was all the rage about 10 years ago. We still believe it's a fairly good visualization, even if it isn't that mobile friendly or up to today's design standards. We used to implement Simile Timelines for all our collections, generating the required XML file from the collection's metadata and customizing the timeline code to work with our collection's time spans. We still proudly use one on our main digital collection website to this day.

However, with CollectionBuilder, we realized it was probably time to say goodbye to Simile Timeline. At first, we thought we might need a sophisticated tool that allowed for unique viewing methods such as horizontal scroll. Yet as we examined other visualizations, we realized a simple table with the capacity to jump to certain years provided the best use case for our collections.

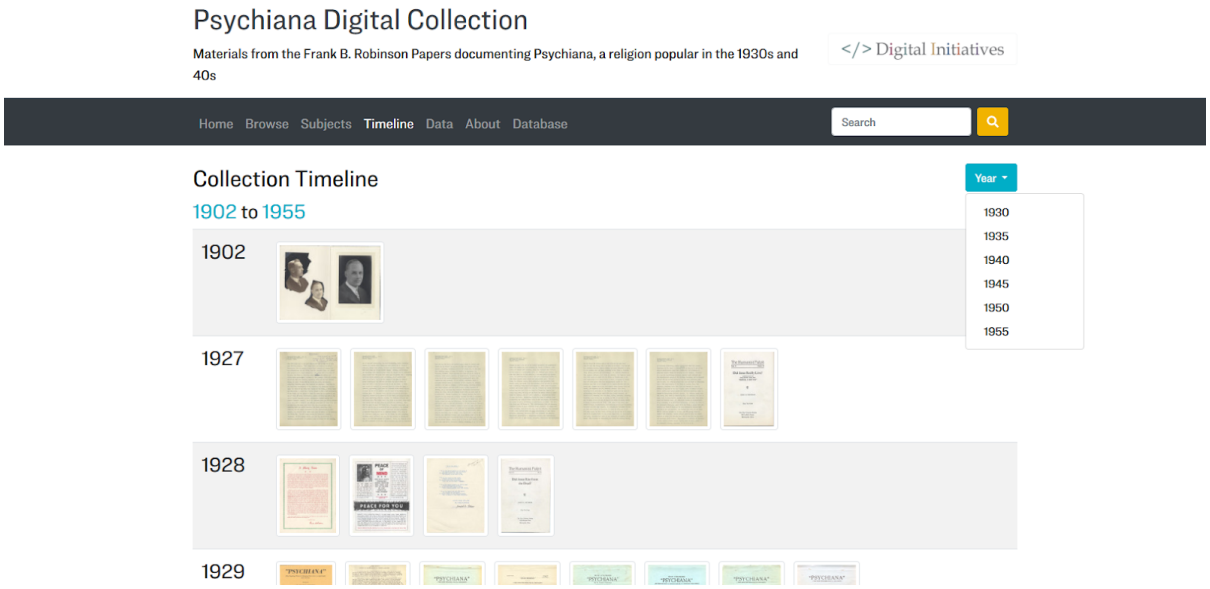


Figure 4. An example CollectionBuilder Timeline page from our Psychiana Digital Collection. A simple table format has proven to be an effective way to organize items by year.

The Liquid code that generates the timeline operates by mapping all the dates in a collection and finding unique years. It then builds a row for each year that drops a thumbnail or text-based card for each item that has a date field containing that year. We have found, somewhat to our surprise, that this is often the best visualization for us to get a good overview of our collection, and we believe it to be a very effective timeline visualization, possibly even better than Simile's.

We also recognize that there is often a need for curated timelines that look a bit fancier and are more enticing to the user. TimelineJS is a well-known visualization built and maintained by the Knightlab. We had previous experience building TimelineJS visualizations by connecting published Google Sheets with metadata, but with CollectionBuilder we decided to generate a JSON file that could be consumed by TimelineJS to build a timeline. This is currently a feature that can be included on a collection's index in the home-infographic layout or separately as its own page. We use this timeline, for example, on the front page of our Dworshak Dam Collection to provide a historical overview of the politics and media surrounding the debate around building a northern Idaho Dam in the 1960s. The include to add the TimelineJS code to any page looks like this: {% include index/timelinejs.html %}. The feature is driven by the timelinejs.json file, which is generated automatically using Liquid.[10]

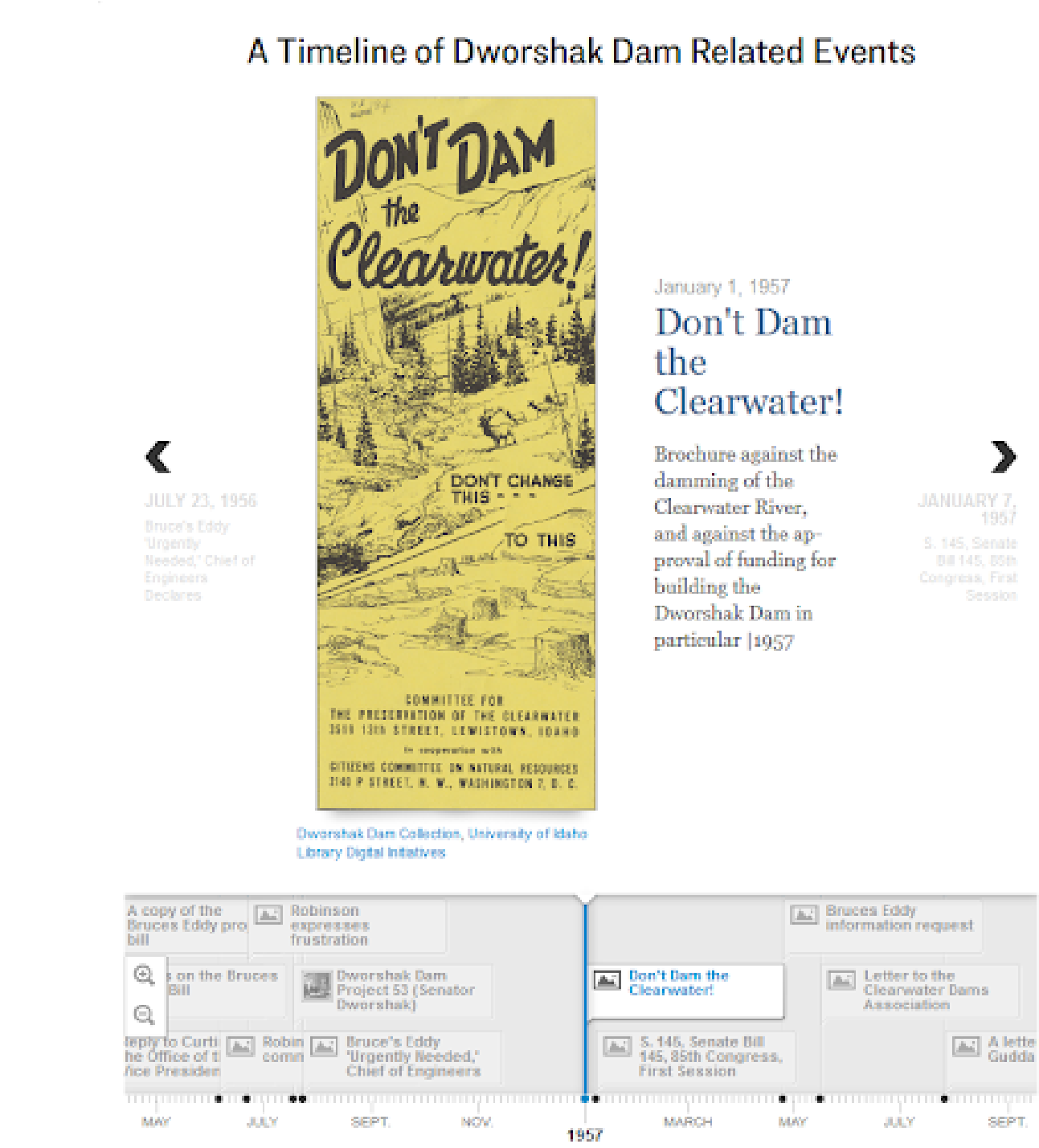


Figure 5. An example of the TimelineJS include on The Dworshak Dam Collection's index page.

Item Pages – Built to be Found

Our earliest skins provided the browsing features described above, but when it came to displaying individual object pages, links would lead directly to the CONTENTdm item page. This was unsatisfying in that it kicked users out of the skin website and into a separate ecosystem without meaningful links back into the discovery features on the collection's skin. This also meant that the representation of the object on CONTENTdm must be the definitive one, featuring the most up-to-date and complete metadata. Given that individual CONTENTdm metadata is notoriously difficult to update in bulk, this can lead to a large amount of work or technical debt when updating aging collections.

The earliest proto-CollectionBuilder site was a project to redesign the Idaho Waters Digital Library. A small grant provided resources to employ a graduate student with disciplinary expertise to enhance existing metadata, as well as add new documents to the collection. This overhaul of the metadata was too difficult to bulk re-ingest into CONTENTdm, so we decided to provide static item pages featuring the enhanced metadata, rather than forcing it back into CONTENTdm. As we further explored this approach, we realized that because static item pages provide the opportunity to: 1) embed rich markup driven directly by a collection's metadata CSV, 2) include links back into the skin visualizations, and 3) customize the presentation of objects, they were a better option for all of our collections.

To generate individual HTML pages directly from the metadata CSV, we turned to the "Jekyll Data Pages Generator" plugin by Adolfo Villafiorita, which was slightly modified for the needs of our project. Configured in _config.yml, this plugin injects a page object for each line of a data file into the Jekyll engine on build.[11] For CollectionBuilder, this means that each object in the metadata will generate an HTML page and all field values will be available to call into the template using Liquid, allowing the metadata to drive the creation of detailed, machine-readable markup and full item representations.

Creating the object pages is handled by the "item" layout. The look and feel of these pages originally mirrored the CONTENTdm item pages, while cleaning up and simplifying the interface with a two column approach (collapsing to one column on mobile). The left side features an image representation and object download buttons that are logically selected based on the format field in the metadata. These use the CONTENTdm APIs to load images and download objects. The right column features the metadata displayed in an easily readable format that can also be copied by the user.

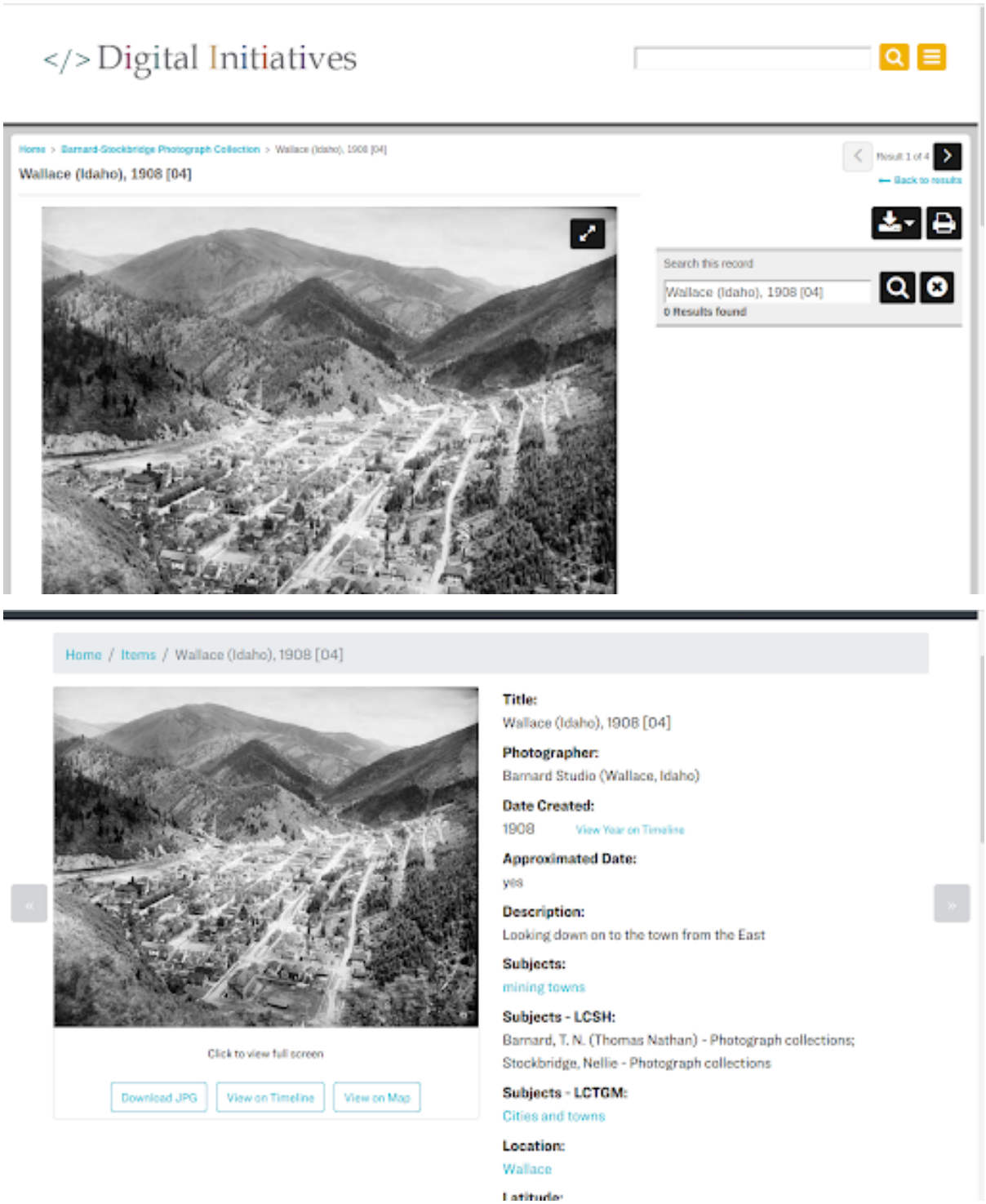


Figure 6. An example CONTENTdm item page from our Barnard Stockbridge Collection (top) and the same item's page in our CollectionBuilder site (bottom).

The metadata fields displayed and meta markup on the item page is configured by the config-metadata.csv, allowing for easy customization of what information is exposed on the page. This config file allows individual fields to be designated as browse links, which transforms values into hyperlinks that lead back to the Browse page, which is filtered on the term. Items with latitude and longitude will feature the button "View on Map," linking to the item's location on the Map page, and likewise those with a date have a "View on Timeline" button, linking to the item's location on the Timeline page. These linkages drive the interactive pathways for the user to explore each collection through visualizations.

Although the item pages may look similar to CONTENTdm's object display, underneath things are quite a bit different. CONTENTdm pages are generated using JavaScript, which loads data from the database, but fails to add any semantic markup standards to the base or rendered page.

Figure 7. An example of CONTENTdm item page markup.

In contrast, CollectionBuilder exposes rich, machine-readable markup driven by the full collection metadata. The Item layout calls a special meta markup in the head element (_includes/head/item-meta.html). This adds three semantic markup standards in addition to standard HTML meta tags.

Figure 8. CollectionBuilder item page, rich meta markup in a variety of standards.

First, this item-meta code adds meta tags containing Dublin Core terms schema based on mappings set up in the config-metadata file. This implementation is based on the output of DSpace repositories (e.g. view the source of this WSU Research Exchange item page). This is not a commonly used form of markup, but since it is used by other repository systems, there may be harvesters designed to crawl it.

Second, it adds Open Graph meta tags. Open Graph protocol was originally designed by Facebook to create a standard for assigning metadata to represent webpages on the social media site. Other platforms, such as Twitter, maintain separate markup standards, but can also read Open Graph as a default, making it a good generic choice to provide this functionality. The Open Graph prefix is declared on the head element:

```
<head prefix="og: http://ogp.me/ns#">
```

The schema can then be used in meta tags following the standard, with values added from the item metadata using Liquid, e.g.

```
<meta property="og:title" content="{{ page.title | escape }}" />
```

This structured data will be used by social media platforms to create the familiar card representations when links are entered into posts, providing an official preview of the content. Thus, if you tweet out a CollectionBuilder item page, the object image or thumb will appear along with the actual item title and description.

Third, the item-meta include creates Schema.org markup in JSON-LD format. The data is contained in a script element of type "application/ld+json." A Liquid template adds appropriate metadata fields, image links, and establishes an "isPartOf" relationship to the full digital collection. Additionally, breadcrumbs are provided in the Schema standard, further reinforcing the contextual relationship of the object. This structured data is used by search engine crawlers to learn more about the page and its relationships, semantic markup that can help with SEO and with the representation displayed to users in search results.

How We Use CollectionBuilder

We started using CollectionBuilder in earnest to develop our digital collections at the end of 2018, using an agile 'sprint' in the Data and Digital Services Department at the University of Idaho Library to jump start the creation of the necessary data and design files. Although this early version of CollectionBuilder was not fully developed, our team was able to prepare the metadata, configuration files, and textual content. Since this data (literally the collection as data) is independent of the CollectionBuilder template, the implementation of visualizations and infrastructure could continue to develop and evolve separately, much like a WordPress theme. Due to staffing issues, we did not return to applying CollectionBuilder to our own collections in earnest again until the summer of 2019, during which we solidified our workflow for building and updating all of our digital collections.

Currently, we use CollectionBuilder via a Git/GitHub focused workflow to develop our digital collections in an open, collaborative way with librarians and staff at the University of Idaho Library. The code for all collections is contained in a single GitHub repository with the master branch representing the generic template. Each individual collection is developed in a new branch within the repository, allowing us to create unique sites while maintaining a central codebase. This approach helps us to keep the underlying code up-to-date with improvements we often make while retaining each collection's individual customizations and commit history that details the changes that were made.

The basic workflow moves through seven areas of activity:

1. **Repository Branch Creation (GitHub)**
   In our CollectionBuilder GitHub repository, we create a branch for the collection, using a branch name that corresponds to that collection's short URL name. For example, the collection https://www.lib.uidaho.edu/digital/barstock/ is developed on a branch named 'barstock.'

2. **Metadata Extraction/Preparation/Revision**
   We extract a TSV file of the collection's metadata from CONTENTdm using the export feature on CONTENTdm's Project Manager tool. We upload this data into Google Sheets, where we check the metadata for errors, correct any field names that need to change (subjects becomes subject, etc.), and add a unique objectid field. Finally, we download the CSV of the document and add it to the _data folder of our branch.

3. **Config File Editing**
   We configure the various files that drive the generation of the site. These include: /_config.yml, where we record which collection and URL we're building; /_data/theme.yml, which controls the basic configurations for all of our visualizations (map, timeline, word clouds, etc.), data files, and our front page; and a series of "config-(…).csv" files (config-map.csv, config-browse.csv, config-metadata.csv, etc.) that further configure what information shows up on which pages.

4. **Development Server Review**
   We serve up the newly configured site on our computer with the Jekyll serve command ('jekyll s'). We then examine the website generated for obvious errors and additional visualization or browsing feature possibilities. We often look at the facets.json data file generated by the system (and configured by the theme.yml file) to see if there might be other fields to visualize in a word cloud format, or even a modified timeline.

5. **Build Static Web Files**
   We've added a rake command to build our website for production. The command 'rake deploy' builds the site for production by running the Jekyll command `JEKYLL_ENV=production jekyll build.` The production environment triggers the addition of features left out during development, such as full meta markup and Analytics snippets. This keeps our analytics account clean from server hits, while allowing us to have the code pre-built with an analytics variable in our _config.yml file.

6. **Deploy Web Files to Production Server**
   We then copy the files and folders contained in the _site folder and add those files to the correct directory on our production web server.

7. **Push Changes to Git Branch of the Repository**
   If more complicated revisions or additions are being made, we try to push changes several times as we revise the site running on the development server. If the changes are small, however, we just push these changes at the end of the process.

When substantial development changes are made to the master branch code, these are pulled into all of the collection branches. These collections are then rebuilt, deployed, and their updated code is pushed to the repository (Steps 5 through 7). This framework facilitates the rapid development of custom visualizations and features, enabling us to generate unique interfaces for more collections. Low priority collections use the standard template, reaping the benefits of the skin interface without a large investment of time. Meanwhile, higher priority collections can be more quickly developed into unique sites, while still maintaining elements of a common theme and branding. For collections requiring extensive customization, the templates act as recipes that can be quickly adapted to new purposes, without getting too far from the original code base, making maintenance easier.

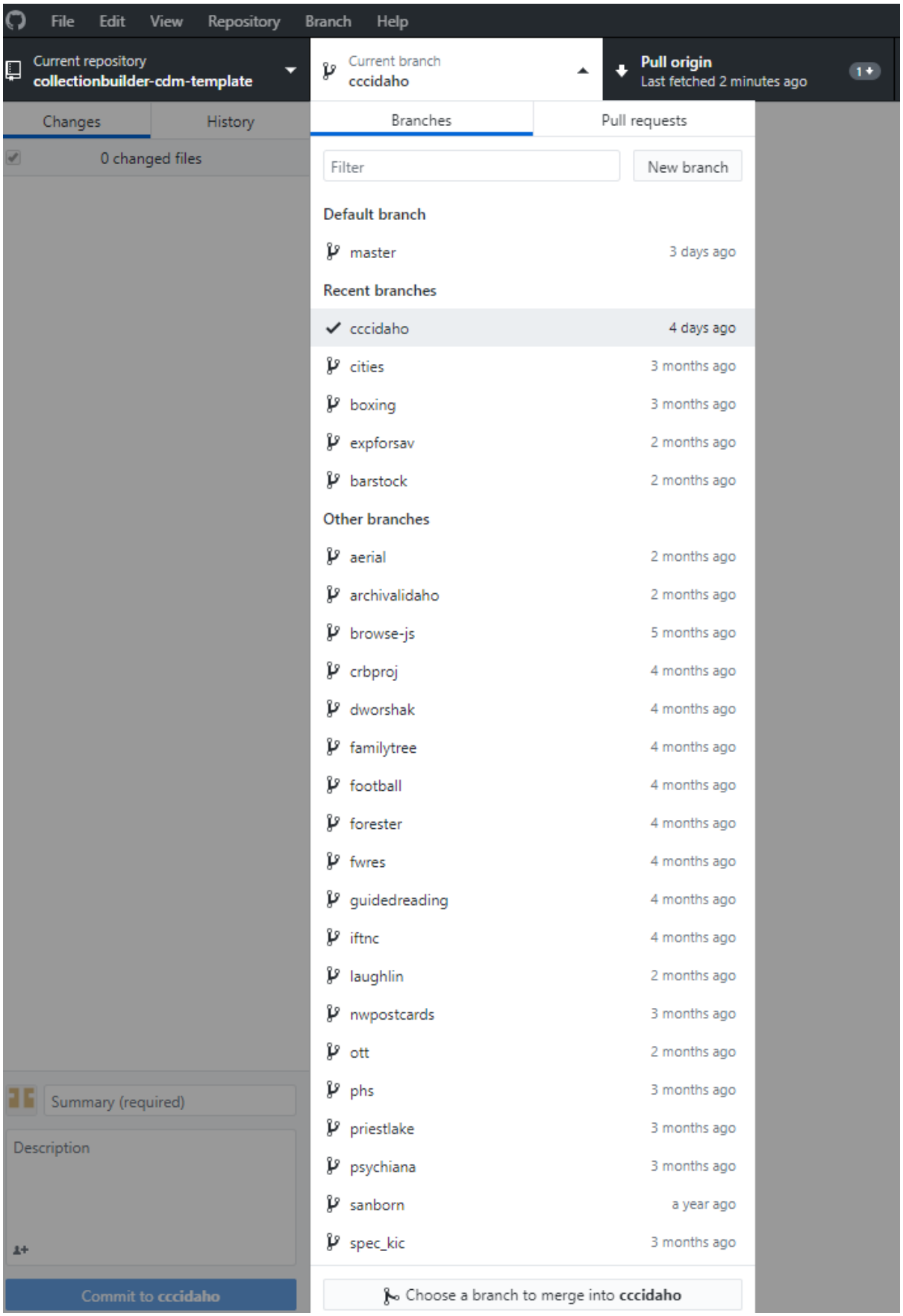


Figure 9. Detail of the branches for our digital collections, as seen listed in a GitHub desktop platform.

Besides being an excellent way to keep our collections' code up to date, creating and managing our collections on GitHub has also expanded collaborative opportunities, allowing more staff to participate in creating collections and giving them an opportunity to learn from the updates and alterations their colleagues make. More people working on collections makes it easier for us to keep our collections up to date, and any mistakes that might be made by those relatively new to working with our digital collections can be easily reverted by CollectionBuilder's developers using Git.

Conclusion: Next Steps

Throughout our development of CollectionBuilder-CONTENTdm, we have been encouraged by the value that this tool can bring to other contexts, especially situations involving teaching digital collections and operating without a DAM system altogether. This has led us to develop other versions of the tool which function independently of the APIs our skin version relies on.

Two versions, CollectionBuilder-GH (GitHub Pages) and CollectionBuilder-SA (Stand Alone), utilize Lunr.js to power searching within collections as fully static sites. GH is designed to be set up entirely using the GitHub web interface and hosted on GitHub Pages. This "lite" version is optimal for pedagogical environments because it does not require users to generate derivatives of their digital objects or install any software, which frees up more time for teaching metadata and web skills. SA combines the optimized code base of CollectionBuilder-CONTENTdm with the independence of GH, allowing users to create a more robust collection not tethered to a DAM system or APIs. This version requires that the user generate thumbnails and small image representations of their full size objects, optimizing the performance and accessibility of the site. Once ready to deploy, these digital objects are hosted alongside the site's static pages on a basic web server of the user's choice.

CollectionBuilder-ES (Elastic) is currently the least developed, but combines the code of the SA version with powerful new search functions that will make large-scale use of CollectionBuilder a possibility. By connecting individual digital collections in a modular fashion, our Elastic version of CollectionBuilder will enable cross-collection searching, and allow for pulling together curated exhibits of items from a variety of collections across different institutions. This could provide a viable alternative to traditional database-driven DAMS systems for individual libraries and organizations, and open unique possibilities to cross institutional boundaries. To accomplish this goal we are working with a developer to integrate Elasticsearch functionality and investigate object storage solutions as well as a variety of deployment options that would enable the tool to follow the JAMstack model.

Ultimately, even as these versions become more mature they will retain their focus on leveraging the collection as data in order to present the collection as a collection, i.e. a grouping that comes to mean more than the sum of its items. Through our continued investment in the metadata driven visualizations that inspired our initial skin version and new features such as our recent efforts to help users more easily compose engaging "about" pages, we endeavor to move beyond the catalog-based view offered by most DAMS platforms. To this end, CollectionBuilder opens up possibilities for the librarians, library staff, archivists, and other GLAM professionals that use it to provide contextually based and data-driven features for the collections they steward. This in turn allows our users to discover and engage with digital collections as collections rather than a series of disembodied items stored in systems we did not design, do not control, and which do not sufficiently communicate the value of the collection as such.

For additional information, please see the CollectionBuilder website, as it is regularly updated with latest features, a variety of documentation, and examples.

About the Authors:

Devin Becker is the Director of the Center for Digital Inquiry and Learning (CDIL) and the Head of Data & Digital Services at the University of Idaho Library. Becker is also a writer. His most recent (static!) web project, CTRL+Shift, provides visualizations and analyses of interviews he conducted with prominent poets across the country.

Olivia Wikle is the Digital Initiatives Librarian at the University of Idaho, where she coordinates the digitization of the University's archival material and builds digital collections that disseminate historical resources. She also works closely with humanities faculty to create digital scholarship projects and teach digital literacy skills to students.

Evan Peter Williamson is the Digital Infrastructure Librarian at the University of Idaho Library, working with Data & Digital Services to bring cool projects, enlightening workshops, and innovative services to life. Despite a background in Art History, Classical Studies, and Archives, his recent focus has been on data-driven, minimal infrastructure web development, currently embodied in the CollectionBuilder project.

Endnotes

[1] This setup could be more colloquially described as a "party in the front, CONTENTdm in the back" approach to development.

[2] Devin Becker and Erin Passehl-Stoddart, "Connecting Historical and Digital Frontiers: Enhancing Access to the Latah County Oral History Collection Utilizing OHMS (Oral History Metadata Synchronizer) and Isotope", code4lib 29, 2015-07-15, https://journal.code4lib.org/articles/10643

[3] The human-readable aspect of this has come as a revelation, as the facets.json file has now become the first thing we look at when designing a new collection. We use it to quickly evaluate which fields have data within them that would reward a word cloud (or visually faceted) expression, as well as to familiarize ourselves with a quick overview of the collection.

[4] e.g. 2016 capture of the Stonebraker Collection

[5] e.g. 2019 capture of Campus Photograph Collection

[6] e.g. Argonaut newspaper headlines

[7] Having thousands of images on the page obviously also requires "lazy loading" to avoid page load issues. To intelligently defer image load, we use lazysizes, a simple to use, up-to-date lazyload library that requires no initialization, and will simply load all images if browser support is missing. Images are given the class="lazyload", and "src" is replaced by "data-src".

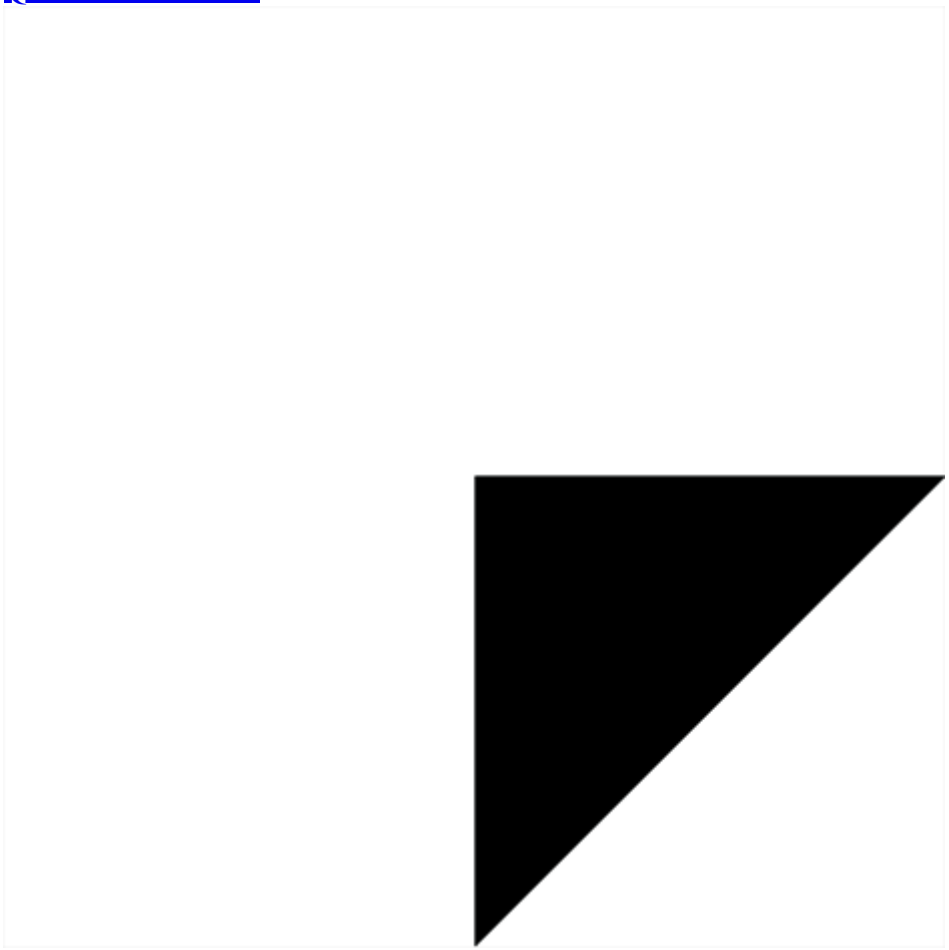
[8] e.g. 2016 Stonebraker Subject Cloud

[9] https://en.wikipedia.org/wiki/Google_Fusion_Tables

[10] We've found that it's best to edit this file down significantly for the best performance, which we do either manually or by creating a truncated CSV of the collection's full metadata CSV file. We then edit the code generating the JSON to refer to that data file, adjusting the "{%- assign items = site.data[site.metadata] -%}" line at the top to reference the truncated CSV, i.e. {%- assign items = site.data.psychiana-select -%}.

[11] Building item pages using our modified version of the Data Page Generator plugin is efficient, since the iteration through the metadata is done in Ruby. However, the sheer number of items becomes a major factor in the build time for the site. Since the Jekyll build process involves writing out so many files on disk, the speed is not limited by your computer's CPU, but by the write speed of your hard drive.

Quinn Dombrowski



Menu

- Home
- CV
- Blog

Home

Sorry for all the Drupal: Reflections on the 3rd anniversary of "Drupal for

Humanists"

Posted by quinn on Fri, 11/08/2019 - 13:27

When I finished writing Drupal for Humanists on July 15, 2015, my Magic-the-Gathering-playing, arithmetic-doing kindergartener was a barely-verbal toddler. The night I finished the manuscript was memorable in more ways than one: I was four months pregnant with my second kid, and it was the first time I felt him kick. When I sent in that manuscript, Donald Trump had announced his presidential campaign just a month earlier, but I paid it no mind as anything but a sideshow, because we all knew there was no way he'd win.

I had no way of knowing that the resulting book would be given a release date of November 8, 2016. By that point, my vague source of nausea while writing Drupal for Humanists had turned into a roly-poly nearly-1-year-old, dressed in a Hillary onesie sent by his great-aunt in Texas. (We waited until late in the election for him to wear it; Berkeley never really got over Bernie.) We believed things were going to be all right with the election, but I joke when I'm nervous. For the months leading up to November 4th, the Day When This Would All Be Over, I'd roll my eyes when I told people the release day of my book, and would quip, "At least one good thing will happen that day!", fully expecting it wouldn't come to that.

You know the rest of the story: my book coming out was, indeed, the only good thing that happened on November 8, 2016. And no one, myself included, cared.

It was an inauspicious start for "Drupal for Humanists". All the things I'd imagined doing to promote the book vanished from my to-do list, replaced by an urgent need to try to wrap my head around what it all meant for us, for our friends and neighbors. I took my oldest kid, in his froggy jacket and rain boots, to an Inauguration Day protest in San Francisco. He rocked a fire truck skirt as he carried a "Refuse Fascism" sign as tall as he was to a rally against the Muslim Ban. On every level, from the climate to the direct threats to one of our favorite preschool teachers, a Dreamer, it felt like the world was starting to unravel.

Drupal wasn't exempt from the zeitgeist. Drupal 8 was released on November 19, 2015 — a year before Drupal for Humanists came out, but after I'd submitted the full manuscript. Chapter 2 includes an extended analogy involving Catalan and Latin, to explain the fracture it caused within the Drupal developer community. I wasn't concerned; it had taken a year for module support to catch up (for existing, widely-used modules — let alone new developments) when Drupal 7 originally

came out, and I expected it would take longer for Drupal 8. That same section included a reference to Backdrop, a fork of Drupal 7 that had been announced in January 2015. At the time, I wasn't impressed: Drupal modules had to be rewritten to remove the database abstraction layer in order to work in Backdrop, and I couldn't see the payoff compared to sticking with Drupal 7 and seeing how things played out. By the time everything else in the world felt like it was going to pieces, it was clear to me that — for digital humanities projects — Drupal 8 had taken a wrong turn. Drupal 8 made it harder, not easier, for the kinds of users I'd written the book for. I wanted website-Legos that anyone with an idea for a DH project could assemble into something very functional and reasonably nice-looking, without writing a line of code. Instead, Drupal 8 was built for Enterprise, where IT teams of developers and sysops folks are paid lots of money to deal with technical processes. As an organization, Drupal was courting developers who were familiar with the enterprise PHP web application framework Symfony, not historians who learned a little PHP while hacking WordPress on the side.

So that was awkward.

I co-taught Drupal at DHSI the summer after the book came out, and my co-instructor, Erica Cavanaugh, taught it the following summer, after my job at UC Berkeley imploded and I was no longer funded to do DH. The Drupal website grew increasingly pushy about Drupal 8, selling a narrative that we knew wouldn't line up with any of our students' experience with using it. But I can't help but wonder about what we sounded like to the students, assuring them that everything was fine, Drupal 7 was still perfectly usable, and in fact the better option, don't worry about starting a project in Drupal 7, all things will be clear by the time Drupal 9 rolls around and all the existing D7 projects have to commit to a direction for the inevitable migration. It was a statement of faith, hope, and incredulity that something as useful as Drupal 7 would just cease to exist.

Time passed, and it became increasingly evident that Drupal, as presented in the book, was a tool for a time and place that I had imagined in 2015, but had not come to pass. Maybe somewhere there's a parallel universe where large-scale communities in the US and beyond have prioritized investment in pragmatic forms of infrastructure that provide technical scaffolding to support digital scholarship, without the building and maintenance burden falling to the scholar. In this one, though, we face more pressing, immediate problems.

Historically, I have not been a fan of the minimal computing approach to web development. A tool like Drupal allows scholars with a much lower level of comfort with technology to build much more complex projects. It is true that the technical

skills that go into the workflow for a GitHub pages Jekyll site (Command-line installs! Markdown! Github! YAML!) are applicable in other contexts. But it is also true that the majority of the scholars I've worked with over the course of 15 years of

doing DH would see learning all that as a very, very lsteep hill — particularly in light of what you actually get at the end. (In 2016, let alone 2019, where there's a big search box in every application and website, explaining that search with Lunr.js is a non-trivial thing to set up isn't going to win you any friends.) Let's be real: it

can be a stretch to ask faculty (especially older faculty) to rethink their materials in a Google Doc to facilitate bulk data import, but it's easier to draw a straight line to how that benefits their project. Getting them to debug the syntax of a YAML file, or learn Markdown for text formatting (instead of just using a familiar WYSIWYG editor) is often too much. Either they'd conclude that the digital humanities was too hard and not worth it, or if they persevered, it would be me — not them — having to handle absolutely every technical thing along the way: the exact opposite of the DIY vision I had for Drupal. (We really wanted to present Wax as an option for image-centric DH projects at this fall's Slavic DH workshop at Princeton, but when we looked at the amount of workshop time we had, and how far that would get students vs. what we could do with Omeka, Omeka won hands-down.)

I continue to struggle with making Jekyll work — even as a person who's generally succeeded in doing "technical" things for some 25 years. I'm still skeptical of how widely it can be adopted, particularly by people who don't have ongoing access to technical support. But I think the biggest change in my thinking over the last seven years (when I first had the idea of writing Drupal for Humanists) has been how much value I place on sustainability, and the inevitability of endings.

When I started Drupal for Humanists, I envisioned a future where things kept getting easier. I imagined a Drupal 8 that included self-updates of the core and module code. Low-cost access to high-quality, responsive hosting services. When researchers felt their project was "finished", it would be trivial to indefinitely maintain what they had built. And when the underlying technical components were all outdated, emulation would save the day! Those were some very wrong prognostications. (Except for the web hosting part — thanks, Reclaim Hosting, for continuing to be wonderful in the face of a general slide towards dystopia!)

The minimal computing advocates were right: database-driven websites with dynamic code are inherently fragile, vulnerable things. They're easy to hack. While high-profile sites with controversial content actively draw the attention of hackers, sometimes sites get hacked for no reason beyond free access to server resources. In Drupal for Humanists, I talked about all the highly configurable, dynamic features that were so easy to implement with Drupal, but never talked about what you lose. It's nice that Drupal has a very granular set of permissions that can be associated with any arbitrary number of "roles" that can be assigned to accounts — but what vulnerabilities do you open yourself up to by allowing people to log into the site to begin with? Making it easy for people to add content to the site is appealing (particularly for directories and the like, where "crowdsourcing" once seemed like a promising model for minimizing labor on the part of the project itself), but so many hacks start with malicious code embedded in such "contributions". If you're

fastidious about updates, and are careful about limiting unvetted users' access to the site, and are using the site regularly to keep an eye on things, you can run a Drupal site for some time without problems. But that's not reality. In reality, things come up: you don't get around to dealing with a critical security update right away, you get sick, you have a family crisis, the grad student you're paying to handle updates gets distracted with comps or struggling with their dissertation, or maybe there's turnover in your local technical support staff and maintenance falls through the cracks for a few weeks or months. And then you may find yourself facing a monstrous clean-up job.

When you're building a Drupal site, it can feel like Legos. But in the medium term and beyond, it's a misleading analogy. Barring the interference of careless pets or children with their hearts set on destruction, the Legos you assemble stay assembled until you choose to take them apart. Instead, building a Drupal site is like buying a pony. It seems like a fun and exciting undertaking, but you quickly discover that ponies require constant feeding and cleaning-up-after. You don't ever get a break — you can't shrug and figure the pony will sort out its own food-and-feces situation while you turn your attention to a new project. If you don't take care of it, and don't find (and probably pay) someone else to take care of it, you can try to give it away — but wise technical staff will balk at the offer of having to take on someone else's Drupal site, particularly if it's been neglected for some time. Once your project is done, if you're not realistically going to devote the ongoing resources required for maintaining it indefinitely, it's time to consider what it will take to "archive" it, shut it down in some orderly manner. In essence, the responsible decision is to euthanize the pony.

Almost all the Drupal sites I've helped build over the years are now dead. A few of them I archived responsibly. I've let more starve to death — sometimes finding someone who'll make the good-intentioned mistake to promise to take care of the site without really understanding what that entails. And then there was the DiRT tool directory, where I tried to do the responsible thing by feeding it to TAPoR (mmmm, delicious, nutritious pony!). But I couldn't bring myself to part with DiRT's bones, and they lingered, giving the impression of a functional site even as it had largely decayed, until the organizational owners of the domain name recently let its renewal slip away. It was a mercy.

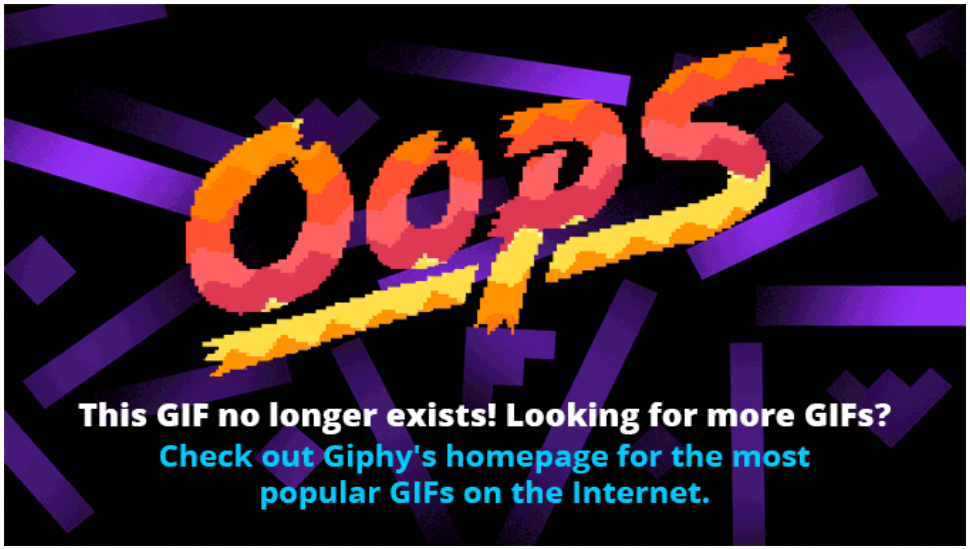
Getting the metaphorical barnyard in order has been a major priority since I started a position doing DH support in the Division of Literatures, Cultures, and Languages at Stanford University, a year ago. Things were in a sorry state when I arrived: my predecessor had built lots of complex Drupal sites, many incorporating custom modules, and he did so in such a way that the scholars were wholly dependent on

him for changes and maintenance. Some of the sites were on servers managed by the digital systems group at the library, who provided system-level updates, but left it to my predecessor to do Drupal updates. Others were on externally-hosted servers, racking up bills of the hundreds or thousands of dollars per year, with account-level credentials that the faculty didn't know. Still others were on free-for-all Stanford-provided webspace, where anyone could run a content management system, provided they dealt with all updates themselves. Two of these sites had been hacked; the one on a library server had been shut down, the one on the commercial hosting persisted in serving malware.

One by one, I've tried to find an approach that will work for these Drupal sites. I was able to do a web archiving capture and data export for one of the hacked sites. Another, I got working again — but we've taken steps towards a more sustainable approach where we'll use library infrastructure to distribute the project's materials in a way that will make them more findable and less siloed. The current CMS will evolve into a static-HTML textbook. I rebuilt a third site more simply, stripping out "features" students were obviously confused by, using a Drupal-based platform operated and maintained by Stanford's Web Services group. Another will probably be migrated to a centrally-maintained university WordPress instance as soon as it launches early next year. See the trend? If I'm going to be in this role for some time, it would take very little time for all my working hours to fill up with managing custom websites, given the faculty expectation that all sites will just continue to work indefinitely. The only way I can ever work on new projects, to support faculty and students' evolving interests, is by getting websites onto infrastructure where someone else will handle the maintenance. This comes with managing expectations about the trade-offs involved (e.g. especially in terms of limiting highly custom functionality), and making the case for why this is the right approach nonetheless.

I'm happy to say that in my first year at Stanford, I have created zero new websites that were not either static, or maintained by Stanford Web Services. (Much love and gratitude to my colleagues in Stanford Web Services. I couldn't do this job without you all.) I've been building community; I've been working with scholars to create, clean, and analyze data sets; I taught a course and published all the materials (including a data set of dresses!) to GitHub. Every project needs a different balance of nimbleness (e.g. adaptability for the next iteration of a course), persistence (e.g. for graduate student collaborators who will be going on the job market in a few years), and publication in forms that can be unambiguously be called done. There aren't enough of me (and, honestly, I'm not by nature organized enough) to adopt the formalized charters and project management workflows used by Princeton's Center for Digital Humanities, which include a "living will" for the project from the start, defined points where developers can accommodate technical changes, and

defined periods for data entry — rather than having to accommodate changes to structure and/or data at any point, indefinitely. But now I try to go into projects advocating that same mentality: it's risky to start a project where you can't articulate what it would look like for the project to be done.



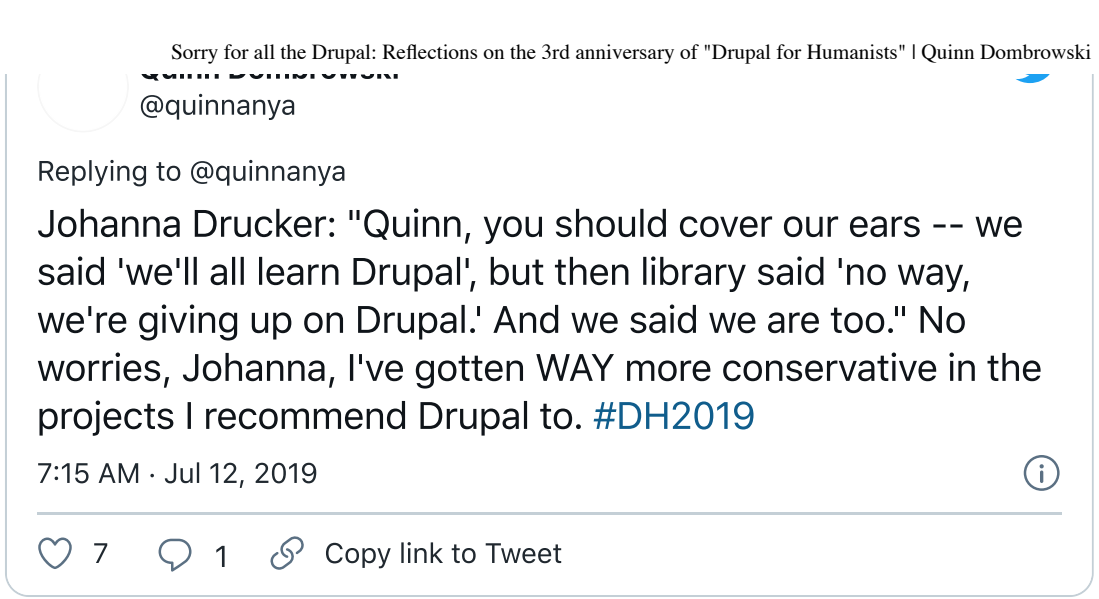
I've realized that Drupal, as depicted in *Drupal for Humanists*, was the kind of DH recommendation that you might get from the Bad Idea Bears in "Avenue Q". Look at all the things you can *just do*! Don't you want to *go do them*? You can structure your data *however you want*! You *don't even need to decide* on what questions you're asking, and *why* — just *put your digital research collection in* and then build some queries and *insights will leap out*! Don't worry about updates, they're not that hard to do! *Yaaayyyy!!* There's been lots of discussion about using digital humanities methods for supporting an argument, vs. exploring a problem space, and I take no issue with the latter. But in most cases, explorers want to find a way home, share their findings, and seek new horizons. When exploration has no exit strategy, it becomes something else.

Sometimes, what people are setting out to do is, in fact, something else — at least for a time. The Modernist Archives Publishing Project does actually want to run a virtual archive that reunites letters, order books, and other digitized ephemera related to the Hogarth Press. These materials are distributed across numerous physical archives (not all of which support IIIF), and having them in one place is useful for this group of scholars and their colleagues. As they have taken ownership

for the site (originally built by my predecessor, who had sole access to admin privileges), they've come to understand what it means to be running infrastructure — and they still want to do it, at least for now. They do actually want a pony. But they're also planning for a future after the pony, accessioning (with permission) the materials they've collected and the metadata they've generated into the Stanford Digital Repository, where it can persist as a unified "collection" in an infrastructure with the goal of long-term preservation, and funding/staffing to match. Other groups, like the Center for Digital Editing at the University of Virginia, have developed and run Drupal-based infrastructure for their own projects like the George Washington Financial Papers, and work with projects with similar needs to adopt and adapt the same approach. They're something like a horse fancier society, breeding and caring for ponies, and helping others do the same. That's also a reasonable approach!

So what is a project like MAPP or a group like the Center for Digital Editing to do when faced with the upcoming cliff of Drupal 7 end-of-life in 2021? In cases where Drupal 7 aligns well with the project's goals — even when factoring in maintenance costs — Backdrop has evolved into what I hoped for from Drupal 8. They stepped back from their earlier, more complicated requirements for porting a Drupal 7 module to Backdrop. They've developed auto-updates for Backdrop core, to decrease the maintenance burden — and what's more, they've incorporated a majority of the most widely-used modules into Backdrop core, so that code can benefit from ongoing automatic updates, too. There's still a few kinks to be worked out; the developer community for Backdrop is smaller than Drupal, and fewer sites are using it, meaning it can take longer for bugs to be identified and fixed. Backdrop's primary audience is non-profits and small companies (i.e. groups with technical skills and staffing levels more comparable to digital humanities projects than large corporations), and uptake has been gradual to date. But I have reason to hope that Backdrop will continue to develop into a viable next step from Drupal 7. By taking the plunge and migrating early, MAPP is doing a service to the broader DH community by funding the port of a DH-oriented Drupal 7 module that they need (Partial Date), which will then be available for anyone else to use. With a few exceptions (blocks / panels / page layout being the major one), Backdrop feels like Drupal 7, overhauled to smooth over many of the annoying quirks inherent to Drupal. On and off over the last year, I've been going through the text of Drupal for Humanists and updating it to reflect the changes in Backdrop; I've posted everything I've written so far on the Drupal for Humanists website.

I was grateful for the lowered lights in the audience, because I think I turned bright red at this point in Johanna Drucker's closing keynote at DH 2019:



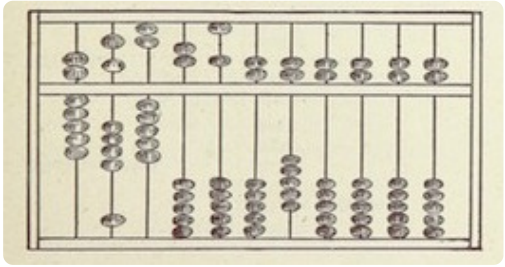
I do think there's a place for Backdrop among the constellations of digital humanities tools and platforms, but it shouldn't be the default. When starting a project, think hard about how long you want to personally be responsible for maintaining infrastructure — not just given your life today, but how it might be different in two years or five years from now. (If I had really been thinking about the consequences of having kids a few years later, I would have done things differently with DiRT and DHCommons.) What are the dependencies for your project to keep running in the form you plan to create it in? What other forms can it meaningfully exist in, in the longer term? How much work will it take to adapt it to that form? Would it be better for you to start the project in that form? What is your exit strategy when the time comes for you to step away? (Your answer should not involve someone else coming along to keep the project going; it's not impossible, but it is highly improbable, even for projects with a large following.)

Don't leap into buying a pony. Think hard before building a Drupal/Backdrop site.

Thanks to Karin Dalziel for nudging me to write this blog post.

Project:
Drupal for Humanists

Minimal Computing



a working group of GO::DH

Home

The User, the Learner and the Machines We Make

by Alex Gil - 21 May 2015

In general we can say that minimal computing is the application of minimalist principles to computing. In reality, though, minimal computing is in the eye of the beholder. A Raspberry Pi could be understood as an example of a minimalist piece of hardware because the creators reduced computing components to what they saw as a bare minimum to achieve simple tasks. The learning curve for using one, though, can be threatening to beginners, and therefore requires more than minimum effort.

On a user interface, on the other hand, eliminating clutter (unnecessary buttons, distracting design, etc) can also be understood to be part of a minimalist approach, making it easier for users to engage. Google's success, for example, may be owed to the reduction of the search function to one box. In order to achieve this feat, though, we estimate that Google uses an enormous amount of code and data in the back end, needing enormous computing power in turn.

I prefer to approach minimal computing around the question "What do we need?" If we do so, our orientations vis-a-vis ease of use, ease of creation, increased access and reductions in computing—and by extension, electricity—become clearer. In this sense, we aim to understand ways of building that could be referred to as "architectures of necessity" as Ernesto Oroza would call them.

Oroza tells us the story of a man who wanted a little bit more space. Here's his story:

> He lived with his mother in a space that was so small that it couldn't legally be considered a house. He expanded into the hallway, built a kitchen and refurbished the bathroom. He changed the status of the property and acquired a title for it. He got his hands on a permit to build on the roof, as he thought about moving out on his own. In order to do this he had to build an exterior stairway. He set to work on the structure indoors and started the paperwork to divide the property. The appearance of an exterior stairway before the process of dividing the house was finished could be considered a violation, and he could be fined or even lose all property rights to the house he had built.
>
> He understood that the description of the house and its parts depends on the cultural understanding that we have of it, that laws depend on this understanding.
>
> Then, what is a stairway? How does one describe it? Could he build a structure in front of his doorway that looks nothing like a stairway but serves the same function? Maybe just objects stacked in such a way that one can climb and descend them? Or an object by Ettore Sottsass, a stack that includes all of Feijóo's books, a Franz West sculpture, anything?

> He decided on a conceptual shortcut: he built the stairway and waited to be fined. In this way, he gained time. The Law demanded that he cease building the stairway until the paperwork needed to divide the property was finalized.
>
> Years went by. He used the unfinished stairway.
>
> What's a finished stairway?

When I ask "what do we need?" I'm asking scholars around the world—librarians, professors, students, cultural workers, independent: What is enough? What's your finished stairway? Needless to say, workers in the humanities have many diverse goals, so we can focus here on what we consider the most important shared one: the renewal, dissemination and preservation of the scholarly record. I take for granted the intersections of our work with the human record writ-large, and the pressing work of scholarly critique of the present, our teaching and the public humanities.

My own posing of the question "what do we need?" comes from an acknowledgement of the hybrid and global future we see being shaped for the scholarly record: parts digital, parts analog. In this new mediatic environment we continue to protect, study and renew the analog, as we attempt to harness the new media in smart, ethical and sustainable ways. For several reasons, this implies learning how to produce, disseminate and preserve digital scholarship ourselves, *without the help we can't get*, even as we fight to build the infrastructures we need at

the intersection of and beyond our libraries and schools. This means that my minimal computing does not stand in as a universal call, but rather as a space for new questions and practices, an injunction to constantly repeat the question, "what do we need?"

Most scholars need to write and make public. That is one of our core activities, the renewal of the scholarly record, and yet, the writing done today using proprietary tools like Microsoft Word or Google Docs, often required by editors, create a disconnect between scholars and the socio-technical mechanisms that are needed to go from the file formats generated by these proprietary applications to a relatively accessible record. As Dennis Tenen and Grant Wythoff put it in "Sustainable Authorship in Plain Text using Pandoc and Markdown,"

> More than causing personal frustration, this reliance on proprietary tools and formats has long-term negative implications for the academic community. In such an environment, journals must outsource typesetting, alienating authors from the material contexts of publication and adding further unnecessary barriers to the unfettered circulation of knowledge.

The culture of "user friendly" interfaces that helped popularize computers for almost three decades now, and which underlines the dominant role of .docx, .pdf and .epub files today, has also led to some basic misunderstandings of what computers can and should do. In the case of writing, the expectation that you should get what you see continues to distance producers from their tools. As with any human tool, we

need to understand computers a bit more intimately if we're going to use them with any degree of critical awareness, and in order to avoid falling into what Matthew Kirschenbaum dubs the "haptic fallacy," or "the belief that electronic objects are immaterial simply because we cannot reach out and touch them." In our case this need comes with some urgency because what has remained invisible or grossly misunderstood to producers of scholarship in certain parts of the world are the material conditions of their own knowledge production—digital and analog—with noxious effects for labor and ecological practices.

In "Sustainable Authorship," Tenen and Wythoff recommend a workflow that goes from the creation of a text to the generation of different file formats for web and print that involves open technology that is relatively easy to learn, to share and to preserve. Minimal computing of the Wythoff and Tenen variety represents then a return to basics that opens up the possibility of understanding small, but more complete "technological stacks" in order to reconnect producers of scholarship to the tools they use. In their case, minimal computing reconciles minimal knowledge with the production of a minimal artifact, without creating necessary friction for the readers. The learning curve may seem steep, though, for a large number of scholars, despite the reassurances and encouragement of those who consider them minimal. Again, we must ask, "what do we need?" Scholars don't strictly "need" to use the minimal approach recommended by Tenen and Wythoff (as in, they are not required to use them by those who promise to take care of the rest). And yet if we do, we are on our way to fulfilling the need to write and publish in sustainable and ethical ways.

Another consequence of reconnecting with our knowledge production is an increased awareness of the cost of scholarly and human memory on the molecular arrangement of the planet. As Stefán Sinclair and others reported on Twitter, in a recent talk at the joint ACH & Canadian DH Conference 2015 in Ottawa, Wendy Chun prompted the audience to "print it out and delete it," citing the ecological cost of storage for digital preservation. Minimal computing shares these concerns, prodding a creative practice that seeks to reduce our impact while achieving our needs. "Print it out and delete it" is a radical answer. We, of course, are concerned about the whole tree, not just the root. The alternative futures implied by Chun's call implies a minimal computing practice where we open up scholarly artifacts for dissemination for a window of time, say one year, then we intentionally shut them down after interested parties have had a chance to "print." The resulting impact on the consumption of paper and other materials would clearly lead to other problems. This is precisely why minimal computing cannot be a set of decisive answers focusing exclusively on the digital, but rather a set of tentative answers and provocations around the hybrid analog/digital ecologies of the world to come.

A purpose dear to us at GO::DH is access. If we believe that we should have a robust scholarly record available to scholars everywhere trough that global library we call the internet, we eventually must agree that the burden of cost should be lifted from the reader. No model we see, though, convinces us it can give vast-scale access to all networked scholars around the world other than the simplest model: producing our own scholarship ourselves. To do so, we may just have to displace the reliance on "user friendly" mechanisms, and learn how to make our own, imperfect as they may be. In the process of learning how to do so, we may also learn how to leverage

institutional and extra-institutional structures for preservation and discovery. But even more importantly, we may yet regain our class consciousness as workers of memory.

What about you? What's your finished stairway?

Bibliography

Kirschenbaum, Matthew. "Materiality and Matter and Stuff: What Electronic Texts Are Made Of."

College & Research Libraries News, Vol 78, No 11 (2017)

Perspectives on the Framework

John E. Russell and Merinda Kaye Hensley

Beyond buttonology

Digital humanities, digital pedagogy, and the ACRL Framework

John E. Russell is digital humanities librarian and associate director for the Center for Humanities and Information at Pennsylvania State University, email: **jer308@psu.edu (mailto:jer308@psu.edu)** , and Merinda Kaye Hensley is associate professor and digital scholarship liaison and instruction librarian at the University of Illinois at Urbana-Champaign, email: **mhensle1@illinois.edu (mailto:mhensle1%40illinois.edu?subject=)**

There is a danger with digital humanities instruction of falling into the trap of buttonology. By buttonology, we do not mean the study of buttons, nor do w intend the derision of August Strindberg, who, in his story "The Isle of the Blessed," coined the word buttonology to mock scholarly pedantry.

Buttonology is, in its simplest terms, software training that surveys different features of an interface in an introductory manner. In a library one-shot, teaching the library discovery system or showing how to perform an advanced search in a database would be buttonology. Knowing how to upload tex into a tool like Voyant does not help researchers think about what texts should be uploaded, how selecting data relates to a research question, or even what constitutes an effective research question. This type of teaching does not encourage critical thinking, yet digital humanities instruction, in our experience, is frequently focused on showing how to use software rather than reflect on the broader context.

There is a growing body of literature on digital humanities instruction in libraries that extends back at least to 2013.[1] Until recently, this literature mostly sidestepped information literacy, focusing on the nature of librarian-faculty classroom collaborations or on teaching digital humanities tools. However, works by Andrea Baer and Krista White have begun to trace connections to ACRL's "Framework for Information Literacy for Higher Education," opening a line of inquiry that helps connect digital humanities work to the instructional mission of our profession and encourages librarians to reconceptualize their approach to teachin by incorporating digital pedagogy.[2]

Digital pedagogy

Digital pedagogy is quickly becoming commonplace among faculty and across disciplines and is often referred to as *critical pedagogical perspective.* Stewart Varner defines digital pedagogy as the act of "creatively and critically incorporat[ing] technology into assignments in ways that truly enhance student engagem and encourage them to confront how technology impacts the work they do."[3]

For all of the literature on digital humanities and libraries, librarians have only just begun exploring their teaching role in the digital humanities. Since this teach role is often tutorial-based, the literature is mostly practical with a focus on how best to present digital tools. However, there are a few examples where librarian are expressing increased dissatisfaction with the limited scope of technology instruction.

In a recent Council on Library Resources (CLIR) paper, Paige Morgan expressed "some frustration with the workshop approach, specifically how it feeds into researchers' desires to learn new tools quickly at the expense of a more thoughtful engagement with the broader methods and questions of digital humanities, including the type of questions digital humanities allows researchers to ask."[4]

In last year's *dh+lib* special issue, Sarah Stanley and Micah Vandegrift described the tool-focused state of digital humanities instruction (both inside and outsid the library), with Stanley specifically arguing that "we should be teaching students resources for working better (both together and alone), rather than what the GUI on different mapping tools looks like."[5] The desire for a more information-literate approach to digital humanities instruction is also the motivation behind Susan Powell and Ningning Nicole Kong's article advocating for an intensive workshop model that "gives librarians the space to move beyond solely skills-bas learning outcomes to more advanced, situated knowledge."[6]

These librarians are expressing a desire to increase their focus on the digital humanities context rather than on software specifics by moving from a skills-base approach to a more conceptual form of teaching. Creating educational experiences with the sole goal of showing how to manipulate software interfaces outsid a larger context is not satisfying to the instructor, and it does not get at the "thoughtful engagement" that Morgan mentions. After all, the most significant barrie digital humanities practice is not how to make the software function, it is the critical engagement with digital methodologies, as well as humanities sources as data, and then organizing data in a manner that allows for subsequent analysis and presentation.

How can we make our digital humanities instruction more information-literate?[7] What might digital pedagogy look like if teaching a session on text analysis or Palladio promoted the kinds of critical reflection as called for by the Framework? We could not find much in this area in the literature, and a quick search in the still-new ACRL Framework Sandbox reveals very few submissions related to the humanities, without a single entry for "digital humanities."

We propose digital humanities instruction should be thought of as a two-step instructional process—adding value to buttonology with a focus on further developing research questions, managing data, and refining methodology. It isn't that skills-based instruction isn't valuable, we know that it is. However, the theories and concepts presented in the Framework align well with the definition of digital pedagogy, especially around concepts of critical reflective practice. As digital humanities projects find their way into the classroom, we are provided with the opportunity to collaborate with faculty to uncover the intersections betwe digital humanities methodologies and information literacy concepts. For example, teaching basic mapping literacy and ethical use of data before tool basics wi prepare learners with the foundational knowledge needed to create a successful map, now and in the future.

Theories of ACRL Framework: Liminality and metacognition

How do we encourage critical thinking so scholars can work towards answers to their complex digital scholarship questions (e.g., how selecting data relates to research question, or even what constitutes an effective research question)? In other words, how do we embrace a critical pedagogical perspective in our digit pedagogy? In addition to the six frames, the Framework outlines several underlying and complementary learning theories—for example, liminal space and metacognition—that can help librarians when designing instruction to go beyond buttonology.

The *liminal state* is the space where learners have begun to commit to the learning process but are consumed with "digression and revisiting." Liminality is not comfortable place for the learner (nor is it, we would argue, for the teacher), but it is necessary in order to move from being a novice to an expert, as summariz in the language of the Framework. Glynis Cousin reminds us as teachers that it is our responsibility to listen for understanding and to nurture a holding environment for the toleration of confusion. "The idea that learners enter into a liminal state in their attempts to grasp certain concepts in their subjects present powerful way of remembering that learning is both affective and cognitive and that it involves identity shifts which can entail troublesome, unsafe journeys."[8] In other words, we are guiding scholars along the process of learning how to learn.

There are specific strategies we can implement to help learners escape the recursive spiral of the liminal state they experience while managing complex digita projects:

- One of the most challenging aspects of teaching digital tools is forgetting what it is like to be a novice learner. Sometimes being a near-novice onese helps you better prepare for the basic problems and frustrations learners are facing. But recognizing liminality is a reminder to you as a teacher that t learning process is not smooth, and it requires anticipating common difficulties and regularly checking in with learners to make sure you are not leavi them behind.

- When meeting with learners one-on-one, make sure to use your in-depth reference interview skills to engage in methods discussions. When a learne in the liminal state, they are not always able to "see the forest for the trees." Your directed questions will illuminate the problems they are having and solutions they had not seen.
- Pay close attention to the digital humanities work and discussions happening on your own campus, as well as across the academic community. Worl through the liminal space may require helping learners make connections to others facing similar problems. Also follow online discussions in order to point your learners to a wide variety of group learning opportunities, such as the active digital humanities community on Slack.[9]
- When designing instructional opportunities, such as workshops and hackathons, pay particular attention to outreach strategies that may bring like-minded learners together, as well as diverse voices. For example, invite the scholar whose project was completed last year to add a more experience voice to the conversation. By encouraging the formation of learning communities on your campus, you are creating safe spaces to help learners navigate the liminal state with others who may be on the other side of struggling with specific digital project issues.
- In designing instructional activities, guide learners through visualization exercises that help to identify "stuck" places. Making graphic representations one's thoughts (e.g., concept maps) can highlight areas that require clarification.

Metacognition, an educational psychology term, is an essential component of the learning process. As defined by Jennifer Livingston, metacognition is "higher order thinking which involves active control over the cognitive processes engaged in learning."[10] For example, if you watch a Lynda.com tutorial on learning Python and afterwards ask yourself how what you learned applies to your digital project goals, that is metacognition in action. Being increasingly aware of your learning is a reflective practice that helps you to solve problems and build self-awareness. Dale Vidmar points out the importance of affect in the instructional process, ". . . [affect] addresses the students' motivation, their involvement in the learning process, their experience of self-actualization and discovery, and the feelings in context of the library environment."[11] The Framework provides illustrations of metacognition and affect through the example dispositions outlined for each frame.

Here are a few specific examples you can apply to your instructional design process to help learners with metacognition:

- Model the metacognitive process during instruction (or in one-on-one consultations) to ask and reflect on big picture questions such as: "What questi can you answer with this tool?" "What can you not do with this tool?" Keep in mind some answers may be simple (e.g., this tool can only work with da in this way, so it is excluded automatically). Also, "Did I get the results I expected? What could I have done differently?" Start with inquiry and build conversations based on the learner's answers. "Is it the data that does not work? Or is the research question fundamentally wrong to begin with?"
- Collaborate with faculty to teach together, modelling your practices while demonstrating a specific tool. This could include thinking aloud as you make decisions so learners can self-correct assumptions. Also, be aware of your own expert bias so you can demonstrate how to clear obstacles.
- Ask learners to specifically define what is difficult for them during the process of instruction. Digital humanities tools are complex and are based on complex methodologies and research questions. By constructing opportunities for learners to self-question as they move from one task to another, th learn to self-assess their progress and adjust accordingly.
- There are several instructional design activities that promote metacognition: think-pair-share, one minute paper ("share a key concept learned" or "wh comes next?"), and case studies.[12]

Conclusion

Digital humanities is all about creating new knowledge and understandings including delving into different ways of thinking in a discipline. In all of the focu on digital humanities projects and on whether or not digital humanities work belongs in libraries, we have lost sight of digital humanities librarianship as a practice of librarianship. There is nothing novel about librarians offering instruction in the classroom or workshop settings, and there is nothing novel about librarians working to connect researchers with the resources they need to conduct research. Furthermore, if information literacy instruction is core to our professional practice, it does not make sense to isolate our digital humanities work from it. In fact, being able to articulate digital humanities work in terms information literacy makes it easier to convey the value of digital work to our peers and administrators.

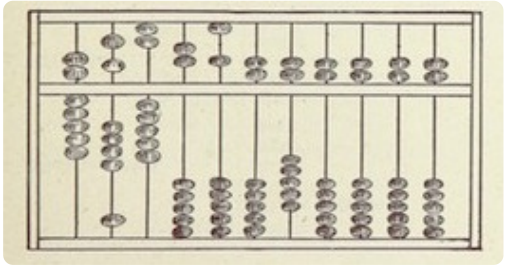
Acknowledgments

This column originated from an ACRL 2017 round table discussion, "Beyond Buttonology: Information Literacy and the Digital Humanities," March 24, 201

Notes

1. Andrea Baer, “Critical Information Literacy in the College Classroom: Exploring Scholarly Knowledge Production through the Digital Humanities,” in *Information Literacy and Social Justice: Radical Professional Praxis,* edited by Lua Gregory and Shana Higgins (Los Angeles: Library Juice Press, 2013): 99–120; John Russell, “Teaching Digital Scholarship in the Library,” *dh+lib,* http://acrl.ala.org/dh/2013/01/24/teaching-digital-scholarship-in-the-library/ (http://acrl.ala.org/dh/2013/01/24/teaching-digital-scholarship-in-the-library/) (accessed October 27, 2017), and “Teaching Digital Scholarsh in the Library: Course Evaluation,” *dh+lib,* http://acrl.ala.org/dh/2013/07/24/teaching-digital-scholarship-in-the-library-course-evaluation/ (http://acrl.ala.org/dh/2013/07/24/teaching-digital-scholarship-in-the-library-course-evaluation/) (accessed October 27, 2017).
2. Baer, “Critical Information Literacy;” Krista White, “Visualizing Oral Histories: A Lab Model Using Multimedia DH to Incorporate ACRL Framework Standards into Liberal Arts Education,” *College & Undergraduate Libraries* 24 (2017): 1–25, http://dx.doi.org/10.1080/10691316.2017.1325722 (http://dx.doi.org/10.1080/10691316.2017.1325722) .
3. Stewart Varner, “Library Instruction for Digital Humanities Pedagogy in Undergraduate Classes,” in *Laying the Foundations: Digital Humanities in Academic Libraries,* edited by John W. White and Heather Gilbert (West Lafayette, IN: Purdue University Press, 2016): 205–222.
4. Hannah Rasmussen, Brian Croxall, and Jessica Otis, “Exploring How and Why Digital Humanities Is Taught in Libraries,” in *A Splendid Torch: Learni and Teaching in Today’s Academic Libraries,* edited by Jodi Reeves Eyre, John C. Maclachlan, and Christa Williford (Washington, D.C.: Council on Library and Information Resources, September 2017): 69–88, https://www.clir.org/pubs/reports/pub174/humanities.pdf (https://www.clir.org/pubs/reports/pub174/humanities.pdf) (accessed October 27, 2017).
5. Sarah Stanley and Micah Vandegrift, “Cross-disciplinarity at the Crossroads,” in *Digital Humanities in the Library/Of the Library: A dh+lib Special Issu* edited by Caitlin Christian-Lamb, et al., http://acrl.ala.org/dh/2016/07/29/cross-disciplinarity-at-the-crossroads/ (http://acrl.ala.org/dh/2016/07/29/cross-disciplinarity-at-the-crossroads/) (accessed October 27, 2017).
6. Susan Powell and Ningning Nicole Kong, “Beyond the One-shot: Intensive Workshops as a Platform for Engaging the Library in Digital Humanities,” *College & Undergraduate Libraries* 24 (2017): 1–16, http://dx.doi.org/10.1080/10691316.2017.1336955 (http://dx.doi.org/10.1080/10691316.2017.1336955) (accessed October 27, 2017).
7. Jan H. F. Meyer and Ray Land, eds., *Overcoming Barriers to Student Understanding: Threshold Concepts and Troublesome Knowledge* (London an New York: Routledge, 2006).
8. Glynis Cousin, “Threshold Concepts, Troublesome Knowledge and Emotional Capital: An Exploration into Learning about Others,” in Meyer and Lan eds., *Overcoming Barriers,* 139.
9. See the Slack Community on digital humanities: https://digitalhumanities.slack.com, HASTAC (Humanities, Arts, Science, and Technology Alliance a Collaboratory): https://www.hastac.org/about-hastac (https://www.hastac.org/about-hastac) , or Digital Humanities Now: http://digitalhumanitiesnow.org.
10. Jennifer A. Livingston, “Metacognition: An Overview,” State University of New York at Buffalo, Graduate School of Education, 1997, http://gse.buffalo.edu/fas/shuell/cep564/metacog.htm (http://gse.buffalo.edu/fas/shuell/cep564/metacog.htm) (accessed October 27, 2017).
11. Dale J. Vidmar, “Affective Change: Integrating Pre-Sessions in the Students’ Classroom Prior to Library Instruction,” *Reference Services Review* 26, 3/4 (1998): 80.
12. For strategies on how to design instruction for technical tools, see Data Carpentry at www.datacarpentry.org and Data Carpentry Instructor Training a http://carpentries.github.io/instructor-training/ (http://carpentries.github.io/instructor-training/) .

Minimal Computing



a working group of GO::DH

Contribute

Home

Minimal Computing in Libraries: Introduction

by Stewart Varner - 15 Jan 2017

[The following paper was delivered at the DLF 2016 Forum in Milwaukee as the introduction to the “Minimal Computing in Libraries: Case Studies and the Case” panel]

Content management systems like Wordpress, Drupal, Omeka and even OJS have had a huge impact in digital scholarship. Before those tools emerged, building a website was actually kind of tough. Even if you managed to get access to a webserver, you had to have a little tech know-how to get something online … and to make it look good was a special challenge.

The rise of the CMS changed most of that and the advent of hosted versions of CMSs changed all of that. No server, no problem. No HTML/CSS skills, no problem. And across the world wide web, millions of websites bloomed …

By and large this has been wonderful. The ease with which you can go from an idea to a project has significantly lowered the barrier of entry for scholars who want to take advantage of the web as a platform for scholarly communication.

However, as we also know, there are no free lunches and the costs of depending on these technologies can be both hidden and steep.

Let's start with free, hosted solutions like wordpress.com. While this is an invaluable tool for some situations, no one would recommend using it for work that needs to be tightly controlled, institutionally branded and preserved for the long term.

This is why many libraries host their own instances of these tools. Of course the decision to adopt these tools has never been made lightly and I bet there are more than a handful of you in this room who have been on either side of those negotiations. The argument adopted by the digital scholarship librarians and others on the public services side was that these tools would allow them to help users get up and running on a project quickly and that it was better, from a preservation perspective, to have most people using the same tool and to have the library (or even the school's central IT office) controlling the sites from the beginning.

On the other side of the meeting table would be sysadmins, the developers and their bosses who were thinking about all the dependencies and the plugins and the themes and all the potential for incompatibility. It's not that these things are impossible to manage but they do take time … time that might be needed to work on the ILS or the repository.

And, of course, everyone was right. It is better to have a good, in-house answer to users who want to build a digital project because, if you don't, you might get the reputation as the place where project ideas go to die. And anyway, some people won't take no for answer. They'll get someone else to build their project, go win a bunch of awards for it and then bring it back to the library because now it needs to be taken care of forever. At the same time, the IT office is not being lazy when they

push back about getting involved with this kind of work. It is time consuming and the maintenance and security costs are ongoing and somewhat unpredictable. I – and others in my role – did not always appreciate the magnitude of what we were asking from the sys admins.

So what we want to talk about today is a kind of third way and it's called minimal computing. I'm going to brag a little and say that I predicted this back in probably 2006. I was using WordPress for the first time and I joked to the person next to me that one day hipsters would rediscover raw HTML and CSS and build artisanal static websites. So, DLF, I'd like for you to meet those hipsters.

While simple static websites is where I have personally engaged with minimal computing, the concept is not limited to web development. Alex Gil, one of our panelists, has written that his engagement with minimal computing hinges on the question "What do we need?" From this perspective - which is the perspective of this panel - a minimal computing approach can speak to web development but also hardware and design.

While it is a simple question, "what do need?," in some ways it marks a departure from what has become business as usual in digital scholarship. From my perspective, it seems that the conversation about digital scholarship has been dominated by a narrative about innovation. However, I've also been sensing a counter narrative emphasizing maintenance and care. This is not only an emerging trend but, in some ways, a reaction against all this talk about innovation.

There are many reasons why you might want to consider minimal computing solutions in your work. Going back to the example of static websites, because there are basically no dependencies, the websites are almost maintenance free. No plugins to update, no PHP to worry about and no proliferating SQL databases taking up space and hiding a bunch of garbage. It might also make it easier to expose your collections as data so that they can be remixed, repurposed and data mined.

If all of that makes you feel selfish about minimal computing, there are also some altruistic reasons to embrace this technology. I enjoy pretty good access to technology as do most people who work in research institutions. However, the way I use the internet is not how everyone on the planet experiences the internet … and honestly, it's now how everyone in my neighborhood uses the internet. Doing our work with a variety of users in mind is an important part of our commitment to access.

As you've probably gathered, i'm excited about the possibilities of minimal computing. However, there are a few issues to keep in mind. First, while "minimal computing" may be small in terms of design and server space, It is not necessarily minimal from a labor perspective. What makes a lot of our unsustainable tools so attractive is that lowered the barriers to creating digital projects. If we raise them again, there will be those who will be at least temporarily left out and it probably won't be wealthy institutions.

Furthermore, there is a temptation when talking about minimal computing to separate "content" from "platform" and "design" as if the these things were

ontologically distinct. One of the great promises of digital scholarship has been that we would have new, exciting ways to share work, tell stories and engage audiences. The platforms are part of how we have done that, they are parts of the stories we tell and our choices about them are meaningful. It should be noted that this problem is not exclusive to minimal computing but I simply want to point out that it has not gone away either.

Literature Geek

Amanda Visconti

About Amanda Visconti
Contact
Download resume
Posts by topic & date
Invited speaking
My DH dissertation
Dissertation posts
Infinite Ulysses

Follow me:

December 08, 2015 *by* Amanda Visconti

Introducing Static Sites for Digital Humanities Projects (why & what are Jekyll, GitHub, etc.?)

I'm working with my university press on new models for DH scholarship... and as I started typing up a description of using Jekyll and GitHub for a proposed DH book project, I ended up discussing not just Jekyll and GitHub but also defining static vs. dynamic sites, git and versioning, and GitHub pages.

Thought I'd share this as a starting point for others! (*Caveat emptor*: *this is one of those emails that turned into a huge email that turned into a blog post.*) If you want to be superfancy, you can click this link to annotate the page with questions and requests for clarification, comments, and suggestions using the magic of Hypothesis, and I'll work on refining this into a more exhaustive FAQ as time allows.

What are.... dynamic versus static websites?

Dynamic sites (like Drupal or WordPress) pull information from a database to populate a page; when you search for some words on Amazon.com, for example, the search results page you are shown didn't already exist as a full HTML page—Amazon.com has a template for search results page with things they all share like their main menu and logo, but it queries the database to insert the results of that search you initiated. Jekyll is a "static site generator" in that it takes page templates (those things like main menus and footers, shared across all the web pages) and other files with specific information (e.g. a file for each blog post on the site) and combines these into full HTML pages for the site visitors to see (i.e. generating a static site, aka a folder of HTML files)—and these are already put together and ready to serve up when you're visiting the site. That is, Jekyll doesn't need to do anything like querying a database when you visit a page; it's already got the pages fully formed, and it just

updates them when and if they ever change. (For someone who took more time to think through a metaphor for static site generation, check out this post!)

How is Jekyll like Drupal/WordPress/Omeka/[other CMS]?

Jekyll is like software called content management systems (CMSs) such as Drupal or WordPress in that it's a set of code that makes a website run and handles certain repeated tasks like displaying a logo and menubar on every page, creating a searchable archive of blog posts, etc. Unlike CMSs, though, Jekyll doesn't have a web "dashboard" (that shiny UI you use to administrate the website: moderating comments, writing a blog post from within the live site...). And! Jekyll does not use a database (the source of a lot of code and security headaches in CMSs like Drupal). If you don't need the power and possibilities behind these CMSs, it's often better for effort, security, and preservation not to use them. Or if you *need* some of the things that Jekyll particularly offers...

Why/when Jekyll?

Versioning! Security (no database hacking) and less sysadminy hassle in general. Speed of loading pages. Free and easy to set up and host with GitHub Pages, and it's then linked into the GitHub.com ecosystem of code versioning, sharing, and reuse.

Jekyll doesn't work as well if you have a site like Amazon.com that basically lets visitors create a huge number of custom pages by making complex searches from a huge set of things, since it's easier for Amazon.com to create search result pages on the fly from their database than to store an HTML page for each possible search result combination. But for a site where you can largely imagine now what all the pages will look like (e.g. a digital book), Jekyll is faster, lighter, and better for longterm security and preservation.

GitHub.com, git, and repos

GitHub.com is a site where people share code they've written so others can reuse it, build on it, and report bugs and feature requests. It's based on git, which is software for versioning code: keeping track of the various changes in code over time as people add, subtract, and otherwise edit that text. People use GitHub.com for things other than code including blogging and professional writing, since it supports keeping track of multiple versions of a text and is friendly to collaboratively authored text. (Other sites besides GitHub use git, and you can use git without GitHub.com. GitHub.com is just a wildly popular public platform for sharing and working on versioned code.)

GitHub "repos" (repositories) are just collections of code; for example, I have one repo with all the code involved in my dissertation, and another repo that shares an old Omeka maintenance-page hack I wrote.

GitHub Pages

GitHub Pages can be located for free at a GitHub subdomain such as amandavisconti.github.io/SGAPedagogyPage (where amandavisconti is my GitHub username and SGAPedagogyPage is the name of the GitHub repository holding its web files). You can also purchase from elsewhere and use your own domain name with the same site (as I did for LiteratureGeek.com). Custom domain names aren't permanently bought, so you need to renew your license over them from time to time (you can do this once a year or buy a bunch of years at a time; the type of domain names a DHer might want to acquire usually cost around $10-20 each per year... except if you want DH.center, they will overcharge you for that!).

You don't need to use GitHub or GitHub pages to use Jekyll; you could, for example, use commercial or university web hosting. I like GitHub Pages hosting because it's free, uses git (which manages multiple versions and drafts of collaboratively authored/edited texts nicely), is set up to make Jekyll use easier, and because GitHub is already the most popular place for DH data sharing, code and text versioning, etc.

Can I use these for projects that might have a commercial aspect (e.g. a book series with digital

components)?

I think so. Jekyll is free and open-source. Its code can be used for commercial projects (its MIT License file with its attribution to the Jekyll creator should be kept with the rest of the files).

GitHub Pages are webpages hosted for free on GitHub.com. GitHub Pages can run Jekyll. My LiteratureGeek.com research blog is an example of a site using Jekyll to run the site and GitHub Pages to host the site. According to this page, commercial use of GitHub Pages is allowed, though because GitHub retains the right to alter that, anyone using it in a book project should have a backup plan in place (something very simple: time required, who to contact, cost of transferring to university or commercial web hosting).

If you do begin to use GitHub for multiple DH projects, you'll want to look into their various plans, as the default free plans are set up for individual use and public repositories (e.g. you might want multiple private repositories so sites can be developed there before going public). There are various options including free plans for educational and non-profit uses.

There you have it, an only slightly-cleaned-up email turned into a blog post! It isn't meant to be a universal resource, but feel free to comment/suggest using Hypothesis via this link.

Recent posts

22 Jan 2020 » Software licensing as feminist & queer digital humanities practice
16 Dec 2019 » Digital humanities job talks: some case studies
02 Dec 2019 » Personal guidelines for DH journal and conference reviewing

What is Static Web and What's it Doing in the Digital Humanities Classroom?

By Olivia Wikle, Evan Williamson and Devin Becker

22 Jun 2020 | 2020 Special Issue

Almost a decade ago, Matthew Kirschenbaum and Micah Vandegrift presented compelling and well-argued ideas about where the locus of digital humanities, or, more broadly, digital humanists should be within the academic context. The intervening years have demonstrated the unique capacity of DH to thrive in a variety of departments, centers, and libraries with specialties that range from making things to theoretical discourse and encompassing everything in between. As the community of DH practitioners has grown, so too has the popularity of several entry-level DH tools. In the classroom context, popular platforms like Omeka and Scalar play an important role in removing barriers and facilitating a relatively easy entry into web authorship for those without coding

skills. New static web-based approaches, however, have emerged as important additions to the DH pedagogical toolbox. These approaches and the tools that facilitate them, such as Ed, Wax, and CollectionBuilder, continue to implement the critical thinking, curation, and storytelling literacies that DH tools teach, while also expanding students' technological literacies into more fundamental areas of computing.

The expanded literacies that these tools encourage include basic knowledge of file systems, web servers, and data management, concepts that students pursuing a humanities-centered education may not typically encounter. Broadening the pedagogical scope to include these concepts provides an opportunity for those teaching DH focused classes and workshops to avoid focusing solely on what John Russell and Merinda Kaye Hensley have termed the "buttonology" of a platform, i.e., teaching specifics of an interface without introducing students to basic technical concepts and methodologies that make the system work. As Dennis Tenen argues, focusing on these broader concepts when introducing a platform makes students less likely to misconstrue the tool itself as a methodology. Such explanations, in turn, help them to avoid the tendency to interpret a project's output as the end goal without trying to understand the hidden algorithms and data manipulation that produces that output.

[W]e should be mindful of how the tools we use fit the contexts in which we teach, and, importantly, how we can use them to encourage both types of learning.

Teaching these fundamental digital skills does not entail a sacrifice: we should not have to give up teaching critical thinking skills in order to incorporate more fundamental computational concepts—part of the uniqueness of DH is its capacity to encourage new ways of thinking via innovative modes of knowledge production. Rather, we should be mindful of how the tools we use fit the contexts in which we teach, and, importantly, how we can use them to encourage both types of learning. Static web approaches, and static site generators, in general, can be used to make explicit the relationship of content as data, which is of both technological *and* critical value to humanities students who are often asked to engage with the question, "What is humanities data?"

Static web tools designed for the DH classroom facilitate teaching fundamental digital literacies because they ask that students use them without the familiarity of a GUI interface. By encouraging students to engage in this exploration in a supported environment, educators can help students learn how to approach digital content with a critical mindset and a nuanced understanding of the systems that control the technology we use, thereby empowering them with a more informed

approach to the digital systems that permeate most aspects of their lives outside the classroom. In a DH context, static web tools have the capacity to reveal rather than hide the computational workings that drive them, promoting hands-on classroom engagement that increases literacies of the web, data, and digital objects.

Static Tools in DH Contexts

In the last decade, dynamic web applications, including content management systems (CMS) such as WordPress and Drupal, have dominated the web landscape as DH platform choices, which often include features such as user authentication, live comments, and endless personalized streams where pages are dynamically rendered on the fly. The functionality that these systems afford, however, comes at significant infrastructure cost, requiring robust server-side processing, databases, and complex software stacks (and the IT expertise necessary to maintain them) to deliver content to users. The details of this complexity are hidden from content creators who interact with the platform only via a web-based administrative interface, positioning learners as software users rather than software authors. This approach fundamentally limits the technical concepts that can be taught, and, as Paige C. Morgan argues, constrains the types of research questions that DH practitioners can ask.

In a DH context, static web tools have the capacity to reveal rather than hide the computational workings that drive them, promoting hands-on classroom engagement that increases literacies of the web, data, and digital objects.

As an alternative to these complex systems, minimalistic approaches powered by modern static web generators have experienced a recent boom. Static site generators are tools that transform a structured folder of files containing content, templates, configuration options, and data to build out a complete website composed of "static" HTML, CSS, and JS files. These generated files can then be copied onto a minimal web server, which requires no database or server-side processing and will deliver the files unchanged to your browser.

In contrast to the dynamically generated pages of CMS platforms, static websites provide several benefits, including:

- faster performance
- lower bandwidth usage

- minimal hosting requirements
- fewer security vulnerabilities
- simple version control

This simplicity also means that static sites are easier to preserve and more sustainable than dynamic sites, as the basic files on the server, even if left unmaintained, will still deliver the website years later, despite the fact that their look may become dated. This is especially important for DH projects given the lack of long term support most DH centers and practitioners can provide for projects. Projects built on CMS platforms, in contrast, are more at risk of becoming malware zombies, a reality that led Quinn Dombrowski to recently caution the DH community not to "leap into buying a pony."

In DH, the use of modern static web tools to build projects is often referred to as minimal computing, which is both a computing practice enacted "under a set of significant constraints," as well as a critical movement that seeks "balance between gains and costs in related areas that include social justice issues and de-manufacturing and reuse." As Alex Gil defines it, the essence of minimal computing is that it attempts to address what a project really needs, using sustainable tools and methods. In practice, minimal computing often entails stripping away unnecessary overhead in order to mitigate reliance on databases and middleware, as well as to relieve significant requirements for processing power and storage.

Gil has been particularly active in developing the concept and enacting the practice of minimal computing. With his collaborator Marri Nyrop, he has developed two "minicomp" tools, Ed and Wax, that serve as excellent examples of the promise of this approach. Both make use of the static web generator Jekyll, as well as GitHub Pages' capacity to host websites from GitHub repositories, acting as templates that facilitate users' entry into the static web within a DH framework. When used in a pedagogical context, as Gil, Nyrop, and others have done in workshops across the country, these projects open up possibilities for students to learn transferable fundamentals of web development and data management that are just as meaningful as the final output itself. In a similar vein, our own project for creating digital collections, CollectionBuilder-GH, is specifically designed to teach both the critical and technical literacies involved in producing digital libraries.

A Scaffolded Approach to DH Literacies

CollectionBuilder is an open source template for creating digital collection and exhibit websites that are driven by metadata and hosted on GitHub Pages. To generate a digital collection, participants:

- create metadata in a spreadsheet
- organize a folder of digital objects
- set up a repository on GitHub
- configure their site's basic settings
- explore their collection website hosted on GitHub pages
- iteratively customize and debug to learn more

The steps to build the collection expand on one another, producing a scaffolded framework that begins with a firm foundation in quality metadata creation and encourages the exploration of new concepts as the collection is developed. The technical and critical skills that emerge from this process encourage the development of interwoven data and web literacies, centered around the collection's metadata as represented within a comma-separated values file (CSV).

By creating well-formed metadata in a spreadsheet, students learn fundamental data (and library!) literacies related to controlled vocabularies, unique identifiers, table-based data representations, and collaborative data cleaning and analysis. As they use these concepts to distill digital archival objects into data in the form of records and fields on a spreadsheet, students also confront the difficulty inherent in curating and representing archival materials online in a way that conveys their original forms and context, making explicit the interpretative biases that necessarily go into this descriptive work. This lesson is further driven home when they see their changes published on the web, which inevitably surfaces anomalies, breakages, and misrepresentations tied to issues in the metadata that they return to the spreadsheet to fix. The iterative nature of this process encourages students to learn the importance of well-structured data and attention to detail, while also helping to demystify "data" in general and complicate the claims often made for its objectivity.

The data literacies students develop in this process are intertwined with several web literacies as well. Students using CollectionBuilder edit and revise their data in a GitHub repository, using Git-based version control. Doing so, they must navigate their repository's directory structure and conceptualize the ways these separate files work together to produce the site. In the process of committing these

> CollectionBuilder's scaffolded nature not only encourages these literacies but also makes the tool flexible enough to be

staged for a variety of learning environments

edits and observing the changes they make, students learn valuable coding, computing, and collaboration concepts that are inherent to version control practices and foundational to modern web development practices. Version control also allows students a safety net to break the code itself, as they can be taught to revert the repository to a former status. This enables them to safely make edits to Markdown, HTML, and CSS files and observe how these edits make a visible impact on the collection site, altering anything from the site's About page to the algorithms producing the visualizations.

CollectionBuilder's scaffolded nature not only encourages these literacies but also makes the tool flexible enough to be staged for a variety of learning environments to focus student engagement in different aspects of the digital collection process. For example, a class of undergraduate History majors at the University of Idaho used CollectionBuilder to create a digital collection using archival materials they curated and digitized themselves. This learning experience prioritized engagement with traditional archival research methods while expanding students' critical understanding of digital repositories and their technical skills. In another scenario, a University of Idaho graduate student created a sophisticated digital collection to complement and expand her dissertation during a summer learning fellowship. In this case, CollectionBuilder provided a new way to think about research data and communicate results. In both of these examples, students integrated data and web literacies with their disciplinary knowledge, employing technical methods that informed and enabled further humanistic inquiry.

Conclusion

Overall, the pedagogical approach we use with CollectionBuilder scaffolds users' learning of open data and web fundamentals via a sequence of tasks that begin with and build off of the simple act of creating a spreadsheet. As this and similar tools (such as Wax and Ed) demonstrate, incorporating static web tools and methodologies into our DH pedagogical practices has the capacity to expand the literacy concepts we teach and to empower students to more critically engage with the digital systems pervasive throughout society.

About the Authors

Olivia Wikle is the Digital Initiatives Librarian at the University of Idaho, where she coordinates the digitization of the University's archival material and builds digital collections that disseminate historical resources. She also works closely with humanities faculty to create digital scholarship projects and teach digital literacy skills to students.

Evan Peter Williamson is the Digital Infrastructure Librarian at the University of Idaho Library, working with Data & Digital Services to bring cool projects, enlightening workshops, and innovative services to life. Despite a background in Art History, Classical Studies, and Archives, his recent focus has been on data-driven, minimal infrastructure web development, currently embodied in the CollectionBuilder project.

Devin Becker is the Director of the Center for Digital Inquiry and Learning (CDIL) and the Head of Data & Digital Services at the University of Idaho Library. Becker is also a writer. His most recent (static!) web project, CTRL+Shift, provides visualizations and analyses of interviews he conducted with prominent poets across the country.

More Posts

« Previous Next »

Using Static Web Technologies and Git-based Workflows to Redesign and Maintain a Library Website (Quickly) with Non-Technical Staff

Evan Peter Williamson, Olivia M. Wikle, Devin Becker, Marco Seiferle-Valencia, Jylisa Doney & Jessica Martinez

2021, University of Idaho Library, Moscow, Idaho, USA

This is an Accepted Manuscript of an article published by Taylor & Francis in **College & Undergraduate Libraries** on 19 Feb 2021, available online:

https://www.tandfonline.com/doi/abs/10.1080/10691316.2021.1887036

Published version citation:

Evan Peter Williamson, Olivia M. Wikle, Devin Becker, Marco Seiferle-Valencia, Jylisa Doney & Jessica Martinez (2021) Using static web technologies and git-based workflows to re-design and maintain a library website (quickly) with non-technical staff, College & Undergraduate Libraries, DOI: 10.1080/10691316.2021.1887036

Using Static Web Technologies and Git-based Workflows to Redesign and Maintain a Library Website (Quickly) with Non-Technical Staff

Abstract

In 2018, a university-wide brand update prompted the University of Idaho Library to re-examine their website development practices and move towards a static web approach that leverages librarian skillsets and provides the library greater control over its systems and data. This case study describes the methodological reasons behind the decision to use the static site generator Jekyll over a Content Management System (CMS) and the practical steps taken to create a sustainable and agile development model. The article details the ways this static web approach (nicknamed "Lib-STATIC") facilitates cross-departmental communication, collaboration, and innovative feature development for library staff members of varying technical abilities.

Keywords: Web development; Git; minimal computing; library web design; librarian workflow design; responsive design

Introduction and Background

During the spring semester of 2018, the University of Idaho redesigned their website and updated their brand, revising the official logos and color schemes (University of Idaho 2018). Like many libraries, the University of Idaho Library independently hosts and maintains its own website, so this university-wide rebranding meant the library website would need a "refresh" as well, to avoid being out of sync with the new look and feel. One might think this would be as easy as swapping out the web banner logo and updating the accent colors (Figures 1 and 2).

Figure 1. The old University of Idaho Library Logo, from https://web.archive.org/web/20180224063150/https://www.lib.uidaho.edu/

Figure 2. The new University of Idaho Library Logo, 1/30/2020.

However, as University of Idaho librarians soon discovered, this cosmetic update was not so simple. Attempts at a quick patch for the new branding were unsatisfying, prompting a long, considered look at the evolving needs of the library's users and the library's overall approach to producing the website. In June 2018, the Library's web team decided that the library needed to completely overhaul its website and development process, and that process needed to be completed by the start of the fall semester in early August. In effect, the university's brand refresh triggered a cascade of change in the technical stack, workflows, and culture behind the library's website—all in a few short months.

An academic library's website plays a crucial role in how it is perceived and utilized by patrons and visitors. Rebuilding that website and establishing a new development style can be an exceptionally difficult process: librarians face an increasingly overwhelming array of choices for deciding on a web platform that fits their context. Investing in an effective platform almost always involves exchanging a certain amount of control over a site's structure and content in order to gain ease of use or convenience. In part, this is a result of libraries lacking dedicated resources and staffing to create and maintain a website full-time. Historically, many libraries

have built websites using content management systems (CMS) that allow for participation from staff without formal web development training, yet are expensive, difficult to customize, and prone to security issues. On the other hand, libraries that forgo a CMS often find that collaborative participation in web design and deployment is restricted only to those with the requisite technical skills.

Not wanting to submit to the lack of control that a CMS requires, yet still desiring an effective means of collaboration, librarians at the University of Idaho Library have developed a modern static web approach for building the library website that offers a viable middle-way. Using the static site generator Jekyll to simplify modular development and the code hosting platform GitHub to facilitate collaboration, librarians produced a complete website composed of static assets without the server-side processing and databases used by dynamic web applications such as CMS. This provided a low-cost, highly customizable, and secure solution with minimal infrastructure requirements.

This case study explores the theoretical reasoning behind implementing a static library website and the practical steps taken to establish an agile yet sustainable development model. Ultimately, the site's creators have found that the most important result of this process is not technical, but social and organizational. The ideals and methods of the static web approach have contributed to an inclusive web development environment in which increased participation empowers librarians and staff to learn and work with basic web development languages and concepts, in turn producing a more robust and uniquely customizable library website.

Static Web Approach to Library Website Development

Deciding Against Using a Content Management System

For many years, the University of Idaho Library website was built using an idiosyncratic PHP-based workflow. At their most efficient, technical tools reflect the cultural needs of the

organization. Here, this workflow was effective for the needs of a lone developer, the Digital Initiatives Librarian (now Head, Data and Digital Services), who completed most editing and maintenance of the site as a team of one. He used a series of HTML templates with PHP includes to create each page. The PHP includes pulled in page elements such as headers, navigation, and footers to ensure the overall theme remained consistent across the site. Although some content was pre-generated from XML data using XSLT, most of the content was manually maintained by editing the HTML files. The full assets of the website were duplicated on a test and production server with PHP installed. All development, testing and backing up was done on the test server that was only accessible to select computers in the library offices, limiting the ability to work remotely or test the site with larger groups.

Although this type of home-grown PHP framework is not unique for generating library websites (Northrup, Cherry, and Darby 2017), it can become cumbersome to maintain at scale and difficult to bring in collaborators with different levels of expertise. Instead, it is far more common for libraries to create websites using a Content Management System (CMS). Research on academic libraries' websites by Connell (2013) revealed that 64% of academic libraries surveyed were using CMS such as Drupal, WordPress, or LibGuides, most often the same platform used by their parent institution. A scan of the fourteen public university libraries in the Orbis Cascade Alliance, University of Idaho's regional academic library consortium, completed in January 2020, demonstrated that this trend has continued. Of the fourteen, 85% use an identifiable CMS (five Drupal, five WordPress, two others), with eight using the same platform as their university, two an older version of the platform, and only four establishing something different (Williamson 2020).

The major CMS platforms such as WordPress and Drupal are complex software that utilize a server-side programming language and database to generate a web-based administrative interface and public facing website. They can offer powerful functionality out of the box

including nuanced user management, ecosystems of plugins to add features, and professional themes. For large organizations, a CMS's ability to establish minute controls over user rights—delegating roles between content editors, web designers, and ITS maintainers—is often especially important. Once established and properly configured, a CMS can enable non-expert users without HTML or CSS skills to rapidly create and edit web content. These upsides have encouraged adoption throughout the last decade and can transform content creation in the context of a library website (Hubble, Murphy, and Perry 2011; Buell and Sandford 2018).

CMS-based functionality, however, comes with high infrastructure costs, requiring powerful servers for their performance, expert developers for their configurations, and IT professionals to maintain system coherence and security over time. Migration from one major version to the next is never trivial, and even routine maintenance necessary to ensure basic site security requires sophisticated, system-specific knowledge to perform and troubleshoot. Role creation and user management can lead to inefficiencies and frictions in workflows with the continual need for role upkeep and assignment. As Yeh et al.'s (2016) research articulates, these common challenges and need for expert support often catch adopters off-guard. In terms of library websites, a CMS may facilitate content creation, but the IT requirements often mean a library must give up significant control of its web pages in order to adopt the university's platform.

For example, the University of Idaho Library has been offered (and declined) the use of the university's proprietary CMS Sitecore, which is managed by the University's web communications unit. Like other CMS products, Sitecore offers the ability to centralize design, branding, and architecture while allowing users from across the university to publish their own content. The result is a coherent, well-branded website, but little of the webpage themes or functionality can be customized by individual units. While the library's web team acknowledges how important it is to mirror the familiar look and feel of the university sites to ensure an

uninterrupted experience for users, the library's website requires more flexibility and agility than the CMS offers. The library's site content and functionality are continuously evolving, reflecting researchers' ever-changing needs to efficiently connect with diverse resources and services.

This independent position offers freedom and opportunity in managing library web properties, but it also requires greater responsibility. In response, the librarians at the core of the web team have developed a pragmatic approach that makes the most of in-house expertise, minimizes infrastructure needs, and respects the unique values of the library, all while ensuring a high-quality experience for users.

Developing a Static Web Approach

To make building and collaborating on a large website possible without a CMS platform, University of Idaho librarians use a modern static web approach powered by the static generator Jekyll and the version control platform GitHub. Static website generators are tools that transform a folder of structured source code into a complete website, building each page as a static asset. The generator works by iterating over source files containing the content, templates, configuration options, and data to build out the HTML, CSS, and JS that make up a website. These generated files can then be copied onto a minimal web server. The static site generator pre-builds all the pages a user might encounter, in contrast to dynamic web CMS platforms that render each page on-the-fly using a database and server-side processing.

Static web generators have experienced a renaissance since around 2015, emerging as a viable alternative for projects of any size due to their simplicity and performance (Biilmann 2015). Combining the power of themes found in CMS with the pure customization of straight HTML, static site generators trade the GUI ease of the CMS platforms for minimal simplicity that provides a more fundamental level of control and the ability to use data to drive content creation. The web infrastructure is simplified, which lowers the IT barriers that databases and

server requirements often impose. At the same time, users interface with the system at a lower level, increasing the difficulty of their initial learning process, but also opening greater opportunities to fully understand the technologies driving the site.

Part of the recent appeal of static generators is driven in response to changing user behavior. As smartphones and mobile data became the norm, user expectations for websites shifted significantly, requiring both responsive designs that function on any size screen and efficient delivery of content at slow connection speeds. Even on the University of Idaho Library website, which features mostly research-related tasks, mobile users continue to steadily rise, from approximately 19% in 2017 to 27% in fall semester 2019 (based on Google Analytics), and bandwidth is a significant concern in a rural state like Idaho. Since static site generators pre-build every page as a static file rather than relying on the server-side processing of CMS, they can provide extremely fast performance, even hosted on the most basic web servers.

Choosing Jekyll as a Static Web Generator

University of Idaho librarians evaluated a wide variety of static site generators, eventually settling on Jekyll for a variety of reasons. First, Jekyll is set up so it supports a simple mental model of how the site will be built that matches up with traditional web development approaches. Static assets in a folder in the source code will become static assets in the same location on the built-out site. Content is represented by stub files that are assigned a layout that pulls together the modular template elements of each web page. This arrangement is similar to the library's earlier templates of PHP includes, built into a tool that makes the approach considerably more powerful and sustainable. University of Idaho librarians' experiences teaching others during classes, workshops, and internal sessions suggest that the biggest barrier to getting started with Jekyll is setting up the development environment, including Ruby, the programming language necessary to run it. Once past that initial hurdle, learners without a development background are able to

understand how the tool works and web pages are constructed. In contrast, some of the major alternatives, such as Hugo, GatsbyJS, and Next.JS, seem to rely on a more formal computational mental model for constructing sites, making them amenable to JavaScript developers, but less intuitive to an average librarian.

Second, Liquid, the templating language used by Jekyll, is powerful yet easy to learn, opening new possibilities for driving content generation from simple data formats such as CSV. This ability to use data created and edited in spreadsheet formats, allows rethinking much of the website content as re-usable chunks added into pages using flexible templates. Spreadsheets are something library folks have plenty of experience with, providing an easy entry point for collaborators to create, organize, and maintain content on the site.

Finally, Jekyll has become the most popular out of the myriad of emerging static generators. This is in part due to being integrated into GitHub's free web hosting service, GitHub Pages, making it an attractive option for quick projects and learning opportunities. The vibrant community around these tools results in better support when encountering issues and a wide ecosystem of quality examples to draw from.

On the surface, “popularity” might seem like a shallow metric to consider when selecting tools, but it has become a significant factor when evaluating the sustainability and usability of different technology choices. In the library’s context, ready availability of quality documentation and help resources can lower the barriers for learning and use. Additionally, tools such as Jekyll, Bootstrap, and GitHub have huge novice user communities that ask questions and post answers across the web. A quick, specific search will almost always return solutions that are comprehensible to non-computer scientists for any issue one encounters. This accessibility of help resources and a community of users is essential to fostering a library-centric approach as well as keeping the workflow "do-able" for University of Idaho librarians and, the authors argue, for librarians generally.

Using Version Control for Better Site Maintenance and Collaboration

Jekyll's connection to GitHub also led to an important improvement in the library's collaborative development practices: the establishment of a version control system and platform. Previous "version control" was manual, i.e., versions were communicated via a series of filenames like index_new.html, index_new-edited.html, and index_better-new-edited.html. This is obviously very prone to error and confusion, leading to an ever-growing maze of filenames and folders. To better manage the history of development (and the Digital Infrastructure Librarian's workload), the library began using the distributed version control system Git with the platform GitHub to host source code repositories. University of Idaho librarians were also attracted to using GitHub because of its emphasis on open code and content sharing, factors that make GitHub popular with librarians at other institutions as well (Davis 2015; Eaton 2018).

When using Git on a project, each set of changes is stored in the repository history as a "commit," like a series of snapshots permanently recording who, what, when, and why. Git also provides the capability of branching and merging, creating an independent copy of the code that can be modified then intelligently re-combined. These features enable collaboration, allowing users to bravely test out new ideas and features without disrupting the current working version or fear losing code.

GitHub provides additional web-based features to facilitate collaboration, which help team members visualize each other's work, track projects, and have conversations directly within the code. By making the team's work visible, GitHub allows the group to better understand what everyone is doing and move forward on the project simultaneously. Finally, the code is available anywhere, allowing collaborators to work outside of office desktops. While a variety of alternative platforms exist, such as Bitbucket and Gitlab or even self-hosted solutions, GitHub seemed to have the most usable web interface, friendly documentation, and largest community, making it an extremely popular repository service and obvious choice for the library's needs.

Building a Template for the Redesign

Several requirements guided the overall project design for the new library web template. The new site needed to:

- follow the University's updated branding guideline for logos, colors, and fonts;
- echo the main university website's look and feel, while maintaining the old library website's unique features, functionality, and structure;
- preserve page locations to avoid broken links;
- and improve responsive design to ensure better usability on all devices.

To build the new template the library web team evaluated a variety of CSS and JS frameworks, which facilitate quick development by providing standardized design components, classes, and functions. The old site used an out-of-date version of Bootstrap 3 with extensive customization and inline styles that made it difficult to maintain. Since Bootstrap continues to be perhaps the most popular framework on the web, the web team decided to update to the most recent version (4.x) and remove the old customizations to ensure simpler maintenance going forward.

Next, the Digital Infrastructure Librarian set up a skeleton structure for the Jekyll project. Using the affordances of the generator, he aimed to create a clear separation of content and design template elements. This not only simplifies maintenance but enables a lower barrier to contributions from collaborators with different skills and expertise. Rather than individual documents, the content is envisioned as data that could be migrated into a variety of templates or platforms, or transformed in bulk, making it future and preservation ready. Additionally, numerous pages presented content in repeating elements on the page such as cards, accordions, or tables. These repeating chunks in the documents can be better represented as tabular data, thus he aimed to move this content into spreadsheets.

Migrating content from the old site was more complex than expected, since the server contained hundreds of files that were no longer in use but lingered for historical reasons. To parse this maze, the Digital Infrastructure Librarian used a web crawler to traverse the website creating a list of pages that were discoverable and active. Using this data, he carried out bulk content migration using Python. Content from each active page was extracted out of the old template by parsing the HTML, cleaned using regular expressions, then exported to a new stub file with the correct format for the Jekyll-based redesign project. This created a raw base for the content, which would need further editing and auditing to ensure everything was up to date.

With the base project source code hosted on GitHub, the initial team of two librarians worked through quick iterations to create the new design, rapidly testing features and styles using Jekyll's built-in development server. Using GitHub Pages hosting, the draft version was published on the live web so it could be reviewed by others and tested on a variety of devices while still in continually active development. Getting feedback early and often is a central feature of an agile approach that helps efficiently direct development efforts. At this point the team of two was about to get bigger, putting the new communication and collaboration workflow to the test.

Effects on Collaboration

Cross-departmental Development using Agile-inspired Development Principles

As the redesign process ramped up, the Head of Data and Digital Services (DDS) department issued an open call to all library employees to join the Library’s annual Web Committee meetings, hoping to gain new members and include as many people as possible in the process, due to the large scope of work to be accomplished. Traditionally, the University of Idaho Library does no large revisions to its website during the academic year, believing that consistency is important for efficient use of the site. This means most of the major revisions and new features

are developed over the summer months when the Library's Web Committee meets and works. Library Web Committee members typically gather feedback and input, open channels of communication, and form working groups to take on new web projects.

For a year prior to the redesign, the DDS department had been experimenting with using Agile-inspired development sprints (https://agilemanifesto.org/) to help improve departmental products, communication, and workflows. That experience, combined with the many constraints presented during the redesign project, led to the initiation of a similar process for the entire Library Web Committee to facilitate the development of the website template and complete migrating the content. Following the Agile sprint model, the committee met every day for two weeks for 15 to 30 minutes in the morning and afternoon. Small groups were assigned specific tasks, and committee members were constantly consulted about the new designs and features being developed each day. Some of the technical work could be accomplished only by the two primary developers but participants of varying skill levels helped with content evaluation and revision, and the editing and migration of some content from the former pages into the new repository.

The informal sprint structure of the process, and iterative development model, allowed many staff and faculty members to provide input on the look and feel of the site as it came into being. The developers were particularly happy to see two reference and instruction librarians, the Science Librarian and Social Sciences Librarian, emerge as leaders in the redesign process. Incorporating public services librarians is integral to any library website redesign for several reasons. First, the website is often students' first interaction with the library and sometimes their only interaction. Since instruction librarians interact regularly with students in both the classroom and the reference desk, they are well suited to identify issues students will likely encounter navigating the website. Because librarians are skilled at the research process and understand a different set of terminology (like resources, services, and collections) than users,

this presents a unique challenge to make the website intuitive. Second, involving more library departments in the design of the website creates buy-in and understanding of website development processes. This makes continual improvement and iteration of the website more feasible (Vassiliadis and Stimatz 2002).

The Science Librarian and Social Sciences Librarian made recommendations for changes to the library's homepage to address issues they and other reference and instruction librarians encountered regularly at the reference desk amongst users. After these initial changes were incorporated into the website redesign, they were then able to take the first iteration of the new website to a new sample audience, running an abbreviated user testing program among typical users, students, and faculty.

Using Student Focus Groups to Gather Feedback

Once the collaborative sprint was completed and the new Library site was prototyped, the Science Librarian and Social Sciences Librarian organized basic user testing focus groups composed of student employees who worked at the circulation desk. Most were upperclassmen who had worked at the library for a few years and therefore not only had experience with the website as students but also in helping patrons utilize it. Pulling focus group participants from the pool of student employees made user testing quick and easy, as they were already in the library and being compensated for their time. The Science Librarian and Social Sciences Librarian sat down with these students to talk about how they used the website and what they wanted to see in the updated version. There were two focus group sessions, each with three student employees. The two librarians first brought up the website as it was and asked them what they thought the main function of the website was, how they found a book, what common questions they got from students and community members about the website, and what frustrations they had with the current website. Their responses detailed some of their frustration

with what department phone numbers were available and where, a desire for the library hours to be in a more prominent location, and requests for the events calendar and the Quicklinks to useful resources to be better highlighted. The students had many thoughts on the old website's design and functionality, ranging from resigned acceptance to commenting that it was "kinda cringey."

The Science Librarian and Social Sciences Librarian then showed the students the updated website and asked about their general impressions, what else they would like to see on the homepage, what caught their eye, how they would navigate the website, and how well the mobile platform worked. The students were very enthusiastic that the new library website design resembled that of the university's main website. This not only kept it in brand but also made it more intuitive for students, who had already learned to navigate the university's site. They liked the consistency in having all buttons be links, the arrangement of items and use of photos, and the prominence of the catalog search bar. One student commented, "Even if this is the final version of the new website, I like it a million times more than the old site."

While these focus groups were helpful in polishing the redesigned site, there was also further value in learning about knowledge gaps of the student employees during this process. Some did not know what a subject liaison was or had never encountered some of the library's more popular databases. This information was useful in designing training for student employees and understanding where knowledge gaps are for students when in instruction or reference situations.

Website Release and Responding to Initial Feedback

Once all the content was edited, the markup formatted to match Bootstrap 4 framework requirements, and the architecture restructured, the new site was built and moved onto the library's server to go live. At this point, the site was functional for University of Idaho students'

and researchers' needs, but there were still smaller projects the team wanted to work on improving over time including gathering wider community feedback on the changes.

To gather community feedback, the Library's Web Committee created a brief, six question Qualtrics survey that was linked to within the initial carousel slide on the library's updated homepage. Within this survey, the committee asked respondents how often they visited the library's website, whether or not they found what they were looking for on the day they responded to the survey, and if anything was confusing or difficult to use on the new website. The committee also included an open-ended question for additional comments. In the span of 7 weeks, the library received 16 responses; 12 of these indicated that they visited the library's website at least a few times a week and, in some cases, almost every day. Overall, respondents indicated that they found what they were looking for on the newly designed website, but six stated that they could not. When prompted for more information, respondents shared that they could not locate a specific database, two specific journals, or a link to Interlibrary Loan (ILL). When comparing the newly designed website (figure 3) to its prior iteration (figure 4), these comments make sense. In the prior iteration, the Library website included links to "Popular" resources, links to find specific types of information, and a link to ILL directly below the catalog search box. These three links were still accessible from the library homepage, but they had been moved to a new Menu navigation box with no obvious signposting.

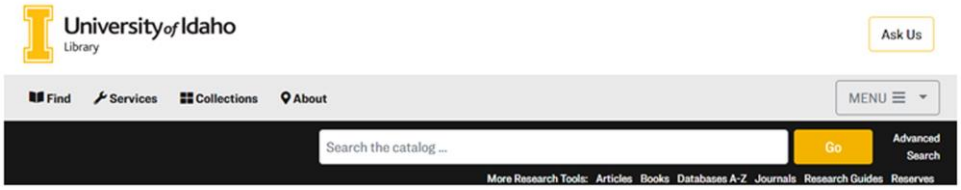


Figure 3. University of Idaho Library website search box, 8/15/2018.

Figure 4. University of Idaho Library website search box, 4/8/2018.

The Library Web Committee also presented the initial website redesign to library faculty and staff across departments for their assessment and found that their feedback on the site's new features mirrored the respondents' feedback from the Qualtrics user survey. Participants in both groups disliked that the search box on the redesigned site now directed visitors to only physical items instead of all the library's physical and electronic holdings and missed having an easy link to Interlibrary Loan situated underneath the search box. Based on this input, the Web Committee released a new version of the library's homepage that incorporated these changes: a link to Interlibrary Loan was added to the "More Research Tools" section below the search box, and visitors searching the catalog would see both physical and electronic results related to their search (figure 5).

Figure 5. University of Idaho Library website search box, 12/15/2018.

Collaborative Development through the Static Web Approach

The capacity for all of the Library's Web Committee members to gather, test, and implement

website design feedback on-the-fly would have been impossible if the library had not migrated from PHP to the static web. Although this migration was challenging for the Library Web Committee and the library, it led to the most successful website redesign to date. Library employees with different skills levels, perspectives, and departmental affiliations were encouraged to share their feedback directly with Library Web Committee members and in open meetings, effectively removing the "us versus them" dichotomy that had dominated prior website work.

The strong channels of communication and collaboration have continued, fostering a greater sense of ownership over web features across the library. This past summer, for instance, reference services meetings discussed the trend of proactive chat boxes to increase engagement with patrons. With her experience on the web redesign, the Science Librarian knew it could be implemented, and sat down with the Digital Infrastructure Librarian to flesh out the concept. In a short time, the feature was deployed throughout the site. This ability to communicate, then rapidly move from idea to concrete prototypes and implementation, is supported by this approach.

Another recent example comes from the development of a "topics of instruction" page that was driven by the library liaisons. The liaisons were interested in marketing the instruction expertise available at the library via the website. A small group of them worked on gathering input regarding expertise via a shared Google Sheet. They also identified possible means of display, noting that the American Library Association's page on future trends (http://www.ala.org/tools/future/trends) was an attractive way of delineating this information. The Liaison to the College of Education, Health, and Human Sciences then met with the Head of Digital and Data Services to collaborate on the project. Through a series of meetings, and then a presentation to the general faculty, the library settled on a page that allows users to filter and search the various areas of expertise (figure 6), learn more about them via modal pop-ups within

the page (figure 7), and then request instruction for the topics they desire via a customized Qualtrics survey form.

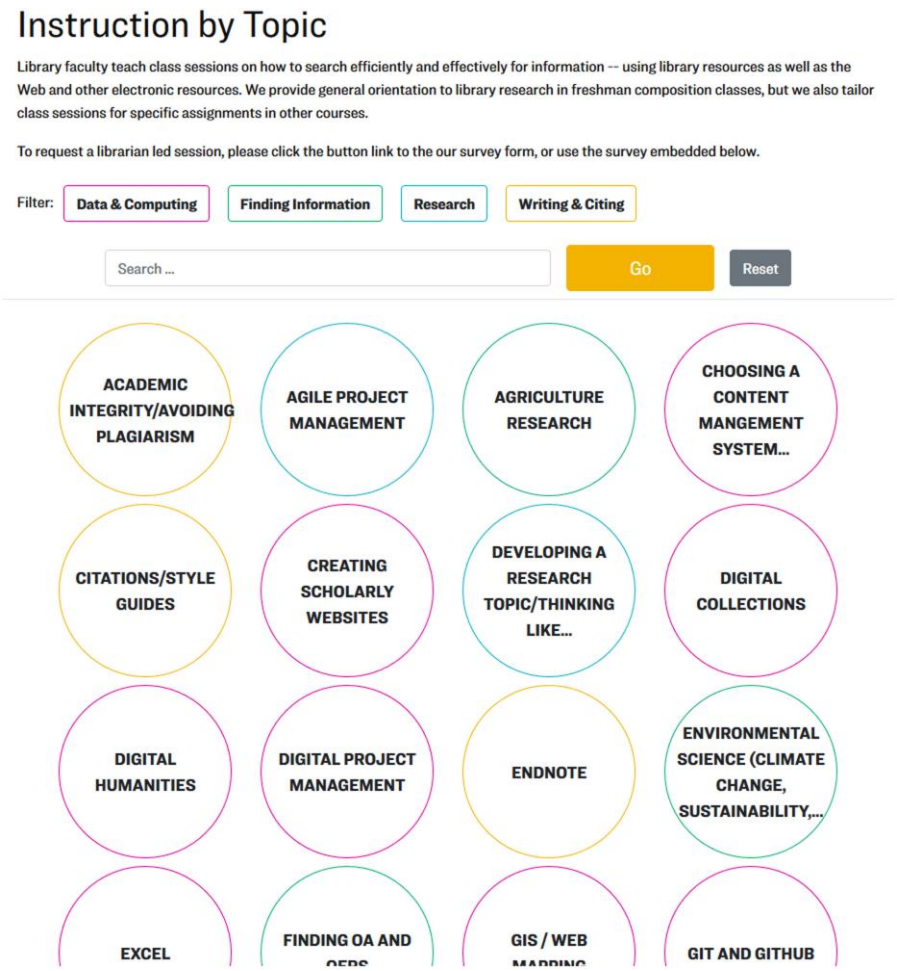


Figure 6. A portion of the University of Idaho Library Topics of Instruction page, 1/30/2020, https://www.lib.uidaho.edu/services/instruction/topics.html

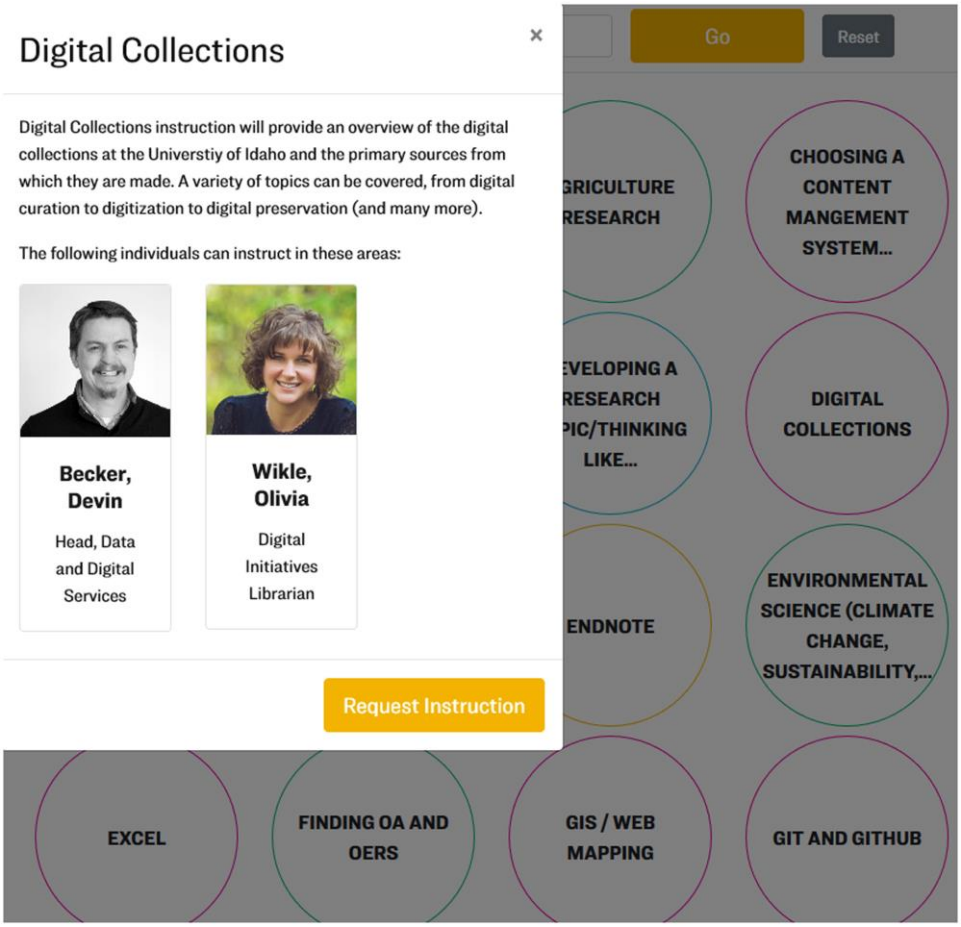


Figure 7. Modal Pop-up for Digital Collections instruction from the University of Idaho Library Topics of Instruction page (1/30/2020). Clicking on the Request Instruction button leads a user to a customized Qualtrics survey form.

To finalize the page, the Liaisons then gathered additional data to better describe the topics listed, via the same Google Sheet. In order to regenerate the page with the new data, the developer downloads the Google spreadsheet as a CSV, replaces the former CSV with that data, then re-builds the site using a Jekyll command, after which he replaces the former page with the revised one. The process demonstrates the agility of this static web approach, as the implementation takes the developer about two minutes while allowing for collaborative content development on a complex web feature using input from librarians across library departments.

Discussion

Benefits and Challenges of the Static Web Approach

The University of Idaho Library's experience implementing a static web approach suggests there are a variety of benefits and unique opportunities in adopting this methodology as an alternative to standard CMS solutions, including minimized infrastructure barriers, project agility, and increased staff collaboration and professional growth.

First, static generators enable simplified infrastructure that requires less technical investment to start and maintain servers and systems. There is no need to configure PHP, update CMS platforms, or maintain a separate staging server with the attendant version control challenges. This also removes significant security risks. The web team had an unfortunate experience where an unused, unpatched WordPress instance was compromised via a plugin that injected advertising into its pages, an incident that made them eager for the peace-of-mind of a fully static server. The minimal hosting requirements allowed a move away from library managed hardware to a basic virtual server managed by central ITS. The static approach enables the library to do more with less IT / sysadmin support, removing technical, budget, and staffing barriers, and refocusing energy on central elements of the site including user experience, site structure, and improved aesthetics.

Second, by focusing on re-usable data and content, this static web approach ensures project agility. This focus stems in part from the web team's experience developing the library's digital collection sites. They saw an opportunity to utilize the data-driven capacities of modern static web to build digital collections sites around spreadsheets of well-crafted metadata and digital objects, both of which can be easily transferred to other platforms in the future.

When it came time to rethink the library website, this focus on data also became the model. Now, the content of the library website is treated as data: many web pages are built from CSVs initially created in Google Sheets, which simplifies updates and collaboration. This

approach has been applied to much of the website and is especially useful when building those pages that include repeating chunks of content, such as directories, resource lists, or FAQs. As with the library's digital collections, organizing the site's content as data ensures that it is prepared for inevitable migration or rebranding projects.

The benefits of this site structure, however, extend further than an increased capacity for migration: because both experienced web team members and more novice library users can easily access and edit this content in its data format, changes can be rapidly deployed to respond quickly to feedback and keep the site up to date. The flexibility of this iterative and data-driven development model encourages experimentation and play by the site's developers, lowering the barriers to implementing new ideas and concepts and enabling incremental improvements to the code.

Finally, besides more in-depth collaboration, this model has created opportunities for colleagues to grow new skills relevant to library work that they might not otherwise encounter. A 2016 survey asking librarians "what technology skill would you like to learn to help you do your job better?" found the top two responses were programming and web skills, which they perceived would help them solve issues, communicate better, and bring new tech into the library (Maceli and Burke 2016). Participating in a static web project provides this opportunity, exposing collaborators to the backend of the website and empowering them to make changes as they develop fundamental code and data skills.

One of the major drivers of silos and work division in academic libraries and similar institutions are the systems employed to deliver library services. By opening up the development workflows, code, and data driving the library website development, library personnel from all areas of the library can develop a deeper understanding and strong sense of ownership over the main access point and consumer of all these systems, the library website. At University of Idaho Library, this new approach has fostered a collaborative spirit that has opened cross-departmental

channels of communication across the library. Because library staff better understand what is possible for altering website content, the web team now enjoys more efficient and rewarding conversations among everyone invested in the website's efficacy and usefulness. Though the learning curve for editing page content is steeper for a static site than a CMS-based approach, once the foundational skills have been mastered, contributors' return on investment includes increased control and customization of site content and their own private development space in which to learn and practice skills.

Lib-STATIC, a methodology

The benefits presented here have potential beyond University of Idaho Library's project and local context. They are part of a growing community of library developers exploring the potential of static development approaches for digital scholarship, digital collections, and web projects of all types. This approach is an emerging methodology that offers a viable alternative to the heavy web infrastructure typically employed in libraries, with the potential to fundamentally reshape librarians' relationship to development. To further support this approach and, most importantly, grow a community of practice around it, the authors have termed this methodology "Lib-STATIC," and have created a website that will start gathering resources, recipes, and ideas: https://lib-static.github.io/ .

At its core, Lib-STATIC recognizes that librarians are fundamentally adept at many of the central needs of online applications—namely metadata/data creation and analysis, content description and classification, and assessment—and seeks to build tools that respect and utilize these skills. Ultimately, this frees librarians from devoting all their time to learning specific proprietary library platforms, allowing them instead to focus their attention on using data and learning web skills that can be applied to lightweight, open source web applications to fit their needs and reflect the unique context and values of the library. This developmental freedom does

have its drawbacks as it requires a greater initial investment of time and energy from librarians learning the tools and techniques, as well as an ongoing responsibility to manage code and dependencies outside the comfortable confines of a CMS. For a certain type of library, the approach, however, offers great potential for librarians' and for library websites. To that end, and to help others learn and employ this approach, the Lib-STATIC website will be built out in the immediate future to feature a network of library projects using static web tools, recipes, and solutions for building sites, and documentation to help others get started.

Conclusion

The University of Idaho Library's static web approach is a user-focused development, design, and deployment strategy that seeks to combine the best of open tools like Jekyll and Git to create easily editable but sophisticated websites. As detailed above, the technical process and project management have moved through several foundational stages of developing this strategy, which culminated in the process used to redesign the University of Idaho Library website in the summer of 2018. While these technical solutions were initially motivated by practical and pragmatic forces related to having only two developers, as well as site security and user experience, this work quickly allowed University of Idaho librarians to discover the potential for web development using a static web-based approach, which the authors have termed Lib-STATIC, as a truly collaborative process that can engage librarians across a variety of technical skill levels.

Lib-STATIC is not a panacea for web development at all libraries, and the approach is more difficult than many GUI-based systems for those first learning the various tools and technologies involved. The methodology, however, provides librarians with a framework for developing and using tools that better embody general library principles of access and usability, while also removing some of the overwrought and expensive systems currently permeating many

libraries. As a community, Lib-STATIC has some distance to go in developing more tools and means for others to implement this development approach, but there is a chance, due to the approach's alignment with the general principles of many librarians, that it will gain some purchase across academic libraries and other GLAM institutions.

Acknowledgments

The authors received support from Institute of Museum and Library Services in National Leadership Grants for Libraries Program award, LG-34-19-0064-19, CollectionBuilder: A Digital Exhibit Platform and Static Web Development Model for Libraries, Built by Librarians (https://www.imls.gov/sites/default/files/grants/lg-34-19-0064-19/proposals/lg-34-19-0064-19-full-proposal.pdf).

References

Biilmann, Matt. 2015. "Why static site generators are the next big thing." *Smashing Magazine*, November 2. https://www.smashingmagazine.com/2015/11/modern-static-website-generators-next-big-thing/ .

Buell, Jesi, and Mark Sandford. 2018. "From Dreamweaver to Drupal: A University Library Website Case Study." *Information Technology and Libraries* 37 (2): 118–26. https://doi.org/10.6017/ital.v37i2.10113 .

Connell, Ruth Sara. 2013. "Content Management Systems: Trends in Academic Libraries." *Information Technology and Libraries* 32 (2): 42–55. https://doi.org/10.6017/ital.v32i2.4632 .

Davis, Robin Camille. 2015. "Git and GitHub for Librarians." *Behavioral & Social Sciences Librarian* 34 (3): 158–164. https://doi.org/10.1080/01639269.2015.1062586 .

Eaton, Mark Edward. 2018. “A Comparative Analysis of the Use of GitHub by Librarians and Non-Librarians.” *Evidence Based Library & Information Practice* 13 (2): 27–47. https://doi.org/10.18438/eblip29291 .

Hubble, Ann, Deborah A. Murphy, and Susan Chesley Perry. 2011. “From Static and Stale to Dynamic and Collaborative: The Drupal Difference.” *Information Technology and Libraries* 30 (4): 190–97. https://doi.org/10.6017/ital.v30i4.1870 .

Maceli, Monica, and John J. Burke. 2016. “Technology Skills in the Workplace: Information Professionals’ Current Use and Future Aspirations.” *Information Technology and Libraries* 35 (4): 35–62. https://doi.org/10.6017/ital.v35i4.9540 .

Northrup, Lori, Ed Cherry, and Della Darby. 2017. “Using Server-Side Include Commands for Subject Web-Page Management: An Alternative to Database-Driven Technologies for the Smaller Academic Library.” *Information Technology and Libraries* 23 (4): 192–97 https://doi.org/10.6017/ital.v23i4.9664 .

University of Idaho. 2018. “Refreshed website, brand resource center launches.” University of Idaho News, June 08. https://www.uidaho.edu/news/news-articles/faculty-staff-news/2018-june/061118-brandresource . (archived: https://perma.cc/B78D-MDSC).

Vassiliadis, Kim, and Lisa R. Stimatz. 2002. “The Instruction Librarian’s Role in Creating a Usable Web Site.” *Reference Services Review* 30 (4): 338–342. https://doi.org/10.1108/00907320210451330 .

Williamson, Evan. 2020. “Library Website Platforms Scan [data set].” Zenodo. http://doi.org/10.5281/zenodo.3653184 .

Yeh, Shea-Tinn, Fernando Reyes, Jeff Rynhart, and Philip Bain. 2016. “Deploying Islandora as a Digital Repository Platform: A Multifaceted Experience at the University of Denver Libraries.” *D-Lib Magazine* 22 (7/8). https://doi.org/10.1045/july2016-yeh .