# From Tamagotchis to Pet Rocks: On Learning to Love Simplicity through the Endings Principles

Martin Holmes <mholmes_at_uvic_dot_ca>, University of Victoria Humanities Computing and Media Centre
Joey Takeda <takeda_at_sfu_dot_ca>, Digital Humanities Innovation Lab, Simon Fraser University

## Abstract

This article, by two of the technical leads on Project Endings, represents the culmination of all we have learned over the last few years, during which we have rescued over a dozen projects from death by software obsolescence and reconstituted them as entirely static, standalone websites with minimal dependencies. We now know a great deal, mostly from our own mistakes, about how not to build robust, long-lasting digital resources, and we have developed a set of principles, practices, and software tools which we believe provide solid defences against digital extinction. Below, we describe the institutional context within which Project Endings was born, and lay out methodically the guiding principles by which we now develop digital editions and other web resources.

## 1. Introduction

This article, by two of the technical leads on Project Endings, represents the culmination of all we have learned over the last few years, during which we have rescued over a dozen projects from death by software obsolescence and reconstituted them as entirely static, standalone websites with minimal dependencies. We now know a great deal, mostly from our own mistakes, about how not to build robust, long-lasting digital resources, and we have developed a set of principles, practices, and software tools which we believe provide solid defences against digital extinction. Below, we describe the institutional context within which Project Endings was born, and lay out methodically the guiding principles by which we now develop digital editions and other web resources.

1

## 2. Early Days

What is now the Humanities Computing and Media Centre (HCMC) at the University of Victoria was once simply the Language Centre, a unit whose role was to maintain language laboratories and computer workstations and to write software applications used for language teaching courses. We had both Macintosh and Windows workstations, and would typically write custom software for either platform but not both, since writing cross-platform software applications was quite difficult in the days before the World Wide Web (WWW) and Java. When the Web first appeared, we seized on it enthusiastically as a practical way to provide language-teaching resources which could be delivered not only on both kinds of workstation in our labs but also to other locations. When JavaScript emerged around 1996, we were blessed with the capability to create truly interactive programmed teaching exercises, and by the turn of the millenium we had built language-teaching exercise collections for many languages, as well as desktop authoring tools to make the creation of such pages easy for non-coders (Arneil and Holmes 1998, 1999, and 2001).

2

The materials we created during this period still exist, and they still work perfectly. Examples can be seen in the Indonesian course materials created in 1999 (https://web.uvic.ca/hrd/indonesian/) and the endearingly-dated "Web Language" online presentation [Holmes 1997], originally created in 1997 and last updated in 1999. The latter includes a feedback form which is still regularly used, so it still clearly has some relevance despite its appearance. The HTML and JavaScript is still fully functional, although non-standards-based cutting-edge technologies of the time, such as RealAudio, no longer work of course.

3
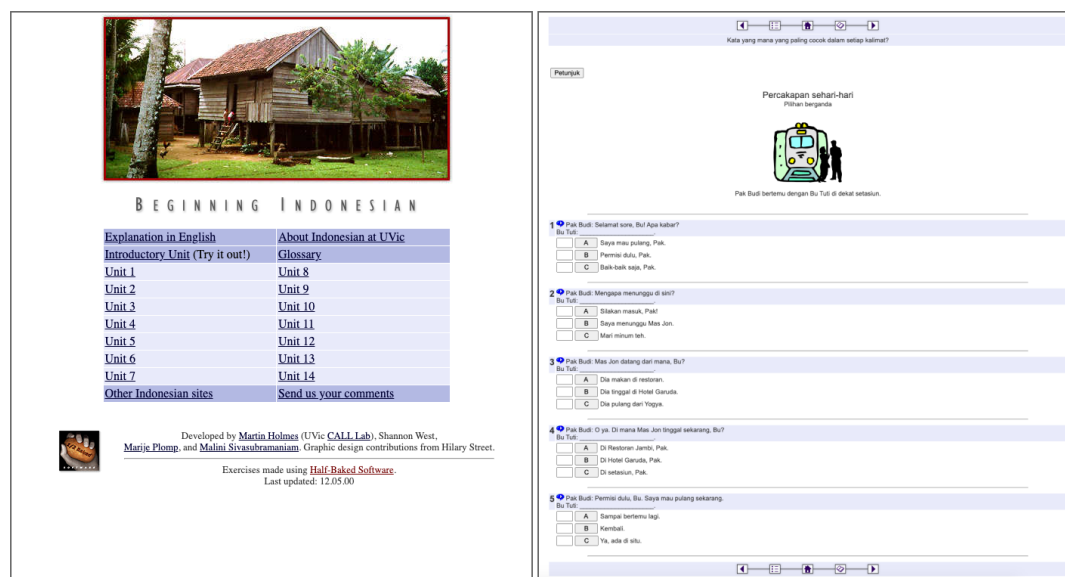
**Figure 1.** Screenshot of Beginning Indonesian homepage (left) and exercise (right). Screenshot taken in 2021, but it looked the same in 1999.

In the early 2000s, the Language Centre was transformed into the Humanities Computing and Media Centre, and given a wider mandate to support what was then commonly termed Humanities Computing, now Digital Humanities. Better-funded projects came along, and more ambitious coding was undertaken, supporting not only language-teaching resources but also digital edition projects, historical datasets, mapping initiatives, and many other research and teaching activities. In particular, we developed some expertise in TEI and the XML-related languages, and began to create large online document collections such as the *Colonial Despatches* project (https://bcgenesis.uvic.ca/) and *Le Mariage sous l'Ancien Régime* (https://mariage.uvic.ca/). In place of static (albeit interactive) web pages, we started to build more sophisticated web applications based on PHP/MySQL database stacks and the eXist XML database.[1] Now blessed with our own server infrastructure thanks to the ground-breaking Canadian TaPOR (Text-Analysis Portal) initiative, we were able to run Tomcat, Cocoon, Postgresql, and a host of other backends to support the growing variety of DH projects under way at our institution.

These server-side systems enabled us (along with the rest of the DH community) to build resources which were more coherent and easier to maintain; information about a single entity, individual, or place could be encoded or stored in a single record, but retrieved and displayed in a multitude of contexts wherever it was needed. Documents could be transcribed and encoded once, but decorated and rendered in many different ways based on the needs of the user or the application context. Sophisticated searching, grouping, and presentation options were easy to implement. Beginning in 2003 we generated a large number of these projects; our servers were well-populated and their CPUs were busy running XQuery, XSLT, PHP, SQL, and even ASP code.

As Lou Burnard (2016) remarks, "nothing in digital form is ever really finished." DH projects may languish untended and unloved for periods of time, but their progenitors typically intend to continue working on them periodically or when funding allows. Setting up such projects implies a commitment to maintaining them, and as our project portfolio burgeoned, so did our maintenance burden. As we moved into the 2010s and onwards, a clear pattern began to emerge: projects created in the 1990s continued to work as intended with virtually no maintenance at all, but the projects we had built on sophisticated backend systems such as MySQL and eXist required frequent and time-consuming maintenance work. As an example, the *Robert Graves Diary* (https://graves.uvic.ca/), one of our earliest digital edition projects, was originally written based on eXist version 0.9 in 2003–2004; it had to be steadily revised throughout 2005–2007, and then completely rewritten for eXist 1.4 in 2012, because the original Tomcat/Cocoon/eXist 0.9 stack became obsolete. Then in 2014 it had to be updated again. This was a project on which the core academic work had been completed in the first phase of the project, but due to infrastructure churn it regularly returned to haunt us. Over the years, we amassed a maintenance burden which threatened to overwhelm our ability to support new projects coming through our doors.

This problem was the impetus for Project Endings. The Web had solved for us the challenge to "write once, run anywhere"; [2] the issue we were now addressing was how to "write once, run forever." Project Endings brought together a team consisting of academic digital humanists, programmers, and librarians, along with a selection of active, complex digital

edition projects, to determine how we could best plan, execute, and archive our projects in such a way that they would live on indefinitely without further maintenance.

## 3. A Catalogue of Failures

In the early days of the Endings project, we began to examine in detail how our older digital editions and sites had progressively broken down. Some problems were obvious: obsolete versions of server-side applications and services (MapServer, MySQL, PHP) had been retired or replaced, and old code would not run on new versions, even if new versions existed. Other failures were more subtle. We investigated, for example, how successfully our applications had been archived by The Internet Archive (https://archive.org/) and reproduced on the Wayback Machine (https://archive.org/web/). It was immediately obvious that sites built on backend databases, where "pages" were in fact queries to the database, were very problematic. The project *Le Mariage sous l'Ancien Régime*, which linked to its anthology documents using URLs that queried the server (e.g. `http://mariage.uvic.ca/xhtml.xq?id=le_bourgeoise_desprit`), was scarcely archived at all, presumably because the archive crawler was unable to follow or parse the query-based links successfully, so for the most part, the archived versions of the site consist mainly of notices that "The Wayback Machine has not archived that URL."

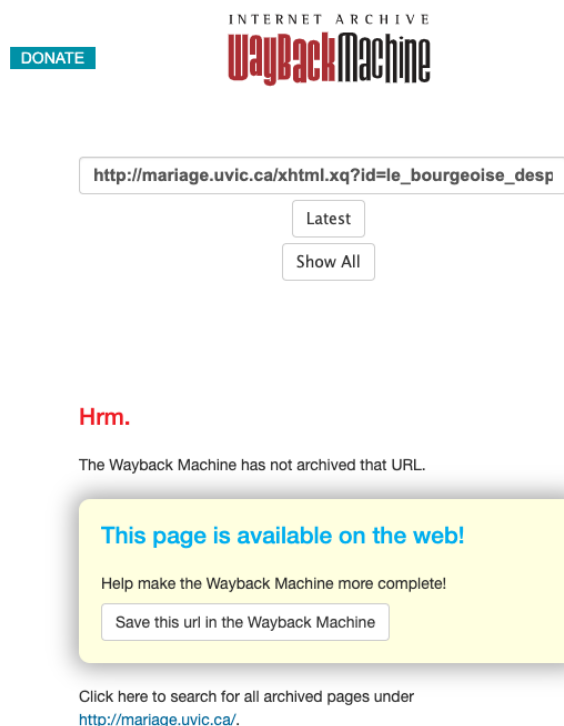<div style="text-align:right">8</div>



**Figure 2.** Wayback Machine error page

*Mariage* also had a rather gratuitous feature whereby the end user could choose between three available colour schemes, and because that functionality relied entirely on JavaScript, the required CSS files had been retrieved and stored as though they were HTML pages, and no styling appeared on the site pages at all. Site search engines failed completely, as did any annotation popups linked from texts which relied on retrieving data by querying the backend database, and JavaScript-based interactivity was generally broken. In fact, the majority of our large-scale, high-profile digital edition projects were scarcely functional at all in the Wayback Machine, and most of their content had not even been retrieved and stored.

<div style="text-align:right">9</div>

The Internet Archive had never been one of our primary backup strategies, but it was disappointing to discover how unsuccessful it had been in its attempts to crawl and reproduce our sites. Although some of these failures could perhaps be laid at the door of the Internet Archive crawler, Heritrix, which had some limitations, most of them were clearly our fault, and resulted from our crude approach to URL construction. One site, the *Robert Graves Diary*, was represented by only a handful of static pages, since the majority of its content was designed to be accessed through explicit searches from the home page, with the search parameters encoded in URL query strings; the crawler would have had no way to know what

<div style="text-align:right">10</div>

pages existed on the site other than the few information and background pages linked directly from the home page. On the live versions of these sites, any pages accessed by a user were typically constructed on the fly by XQuery and XSLT, so they did not exist in any form which could be backed up even locally by any normal method.

Digging deeper into methods for backing up and replicating our projects within our own institution revealed other problems. Some sites and applications included hard-coded links to local resources using absolute URLs rather than relative ones, so if the site were replicated in a different location, it would still be dependent on, and prone to send the user over to, the original site.

11

Another problem faced by projects backed by online databases was a lack of consistency in versioning of content. Typically, multiple project members had rights to upload new material directly into the live database, thus "publishing" it, but there was no attempt to ensure that the current state of the live site was — in Endings parlance — "coherent, consistent, and complete" (see [Holmes & Takeda 2019b]). Someone editing a primary source text might upload a new chapter to the site, forgetting that it contained links to newly-added people in the personography or locations in the placeography which also needed to be uploaded, so those links would fail. Every "page" might have a different publication date, and the site overall had no version or edition information, so anyone citing it would be forced to fall back on the rather unfortunate "last accessed" date to anchor their citation, knowing that the specific version of the document they were citing might change tomorrow with no warning and no way to retrieve the previous version. This was a problem of project organization rather than a technical issue, and it arose largely out of the working methods emerging in the early 2000s. In the excitement provoked by the new affordances and publication possibilities inherent in digital projects, there was a tendency to publish anything and everything as quickly as possible. A measured and thoughtful approach to "rolling publication" is of course possible, but, as we discuss in section 5.5, our work on Endings has convinced us that for most projects, a more traditional edition-based methodology is better.

12

Finally, we were forced to acknowledge that few of our projects had adequate documentation, making remediation and reconstruction doubly difficult.

13

We had, it seemed, spent a dozen years enthusiastically generating fragile, unmanageable, unmaintainable, unarchivable contraptions, and we were going to pay the price for our thoughtlessness.

14

This is not an unusual situation. In fact, our research suggests that it is the norm for DH projects.[3] Many DH web applications are built rapidly, using off-the-shelf frameworks and libraries which date quickly and are ultimately guaranteed to fail, and most projects either rebuild their applications in repeated cycles of funding, or see them languish and ultimately disappear. Our situation was perhaps more pointed and critical because we are a base-funded unit with a long lifetime, committed to maintaining our projects even after their initial funding is exhausted. Units like ours are still relatively rare; in most cases, there is little or no institutional help and no expectation of any. But having caused the problem, we now had to fix it, not just by rebuilding multiple large projects, but by doing so in such a way that we would not have to do the same work again in another ten years.

15

## 4. The Endings Project

The Endings Project was set up as a formal collaboration between the HCMC programmers grappling with the issue of project obsolescence, four Principal Investigators leading some of our largest and longest-running active projects, and a group from the University Library that specialized in digital preservation and collections. We began with the aim of researching, formalizing, and testing a range of strategies for rewriting four projects — *The Map of Early Modern London*, *Le Mariage sous L'Ancien Régime*, the *Nxaʔamxcín Dictionary Database*, and *The Robert Graves Diary* — with longevity in mind. The project, formally titled Endings: Concluding, Archiving, and Preserving Digital Projects for Long-Term Usability, was funded by the Social Sciences and Humanities Research Council of Canada (SSHRC) for four years starting in April 2016. Since then, we have added many more projects, some new and some dating back many years, to the project portfolio, and there are now 14 projects under the Endings umbrella. Each individual project has provided its own unique set of challenges and resulted in fresh ideas and strategies as we have learned to build static sites from them.

16

The Endings project has produced a range of outputs, including a survey with follow-up interviews, software, many articles and presentations, many static-site digital editions,[4] and the 2021 symposium from which this special issue was born. But perhaps the most important outcome of our work has been the set of principles that codifies our approach to site-building.

17

# 5. Endings Principles, or: from Tamagotchis to Pet Rocks

Readers of a certain age may remember the 1990s craze for the Tamagotchi, an electronic device that was designed to mimic a pet animal by requiring regular attention — feeding, education, play — from its owner in order to keep it healthy.[5] Our digital edition projects had become rather like Tamagotchis in requiring constant tending simply to keep them functional. Slightly older readers, however, may remember a similar, pre-digital fad from the 1980s. The Pet Rock was exactly what it purported to be: a piece of rock, which required nothing whatsoever from its owner. "A rock would not need to be fed, walked, bathed, or groomed, and it would not die, become sick, or be disobedient" (Wikipedia). Our Endings mission was to devise methods for turning our Tamagotchis into Pet Rocks.

<div style="text-align:right">18</div>

The Endings Principles present a set of simple guidelines for this process, and are intended for two audiences: researchers who have Tamagotchi projects that are already too needy, and who are looking for an exit strategy; and (more importantly) researchers in the early stages of planning and setting up new projects, who are in a position to build a sustainable, low-maintenance digital edition from the outset.

<div style="text-align:right">19</div>

This article lays out and discusses version 2.2 of the Endings Principles. This version does not differ substantially from the initial draft, first published on the Endings website in 2018, but the major sections have been re-ordered. Those sections cover the five principal aspects of a digital edition project which we consider to be crucial in designing for longevity: Data, Documentation, Processing, Products, and Release Management.

<div style="text-align:right">20</div>

## 5.1 Data

Data is the expression of the source information, knowledge, and expertise of our researchers, and forms the basis and "necessary context for analytical claims" (Thieberger in this issue, and [Thieberger 2016]). Digital Humanities praxis has often tended to blur the distinction between data and products; according to Barats, Schafer, and Fickers, for example, "data management is not just about the data that are used, but also about those, which are produced (metadata, results of research, etc.). In fact, data management covers the whole life cycle of a research project, from data selection to their curation and description, from analyse and interpreting data to its publication and long term storage" para 4. For the purpose of Endings principles, our definition of data is much more constrained: we define data as the source material from which end-user products are generated. In the case of a digital edition, data may take the form of TEI XML encodings of primary source or born-digital documents, but it may also include relational database data, digital images, audio or video recordings, and any other material acquired or generated during research, but which does not form part of the end product.

<div style="text-align:right">21</div>

### 5.1.1 Data is stored only in formats which conform to open standards and which are amenable to processing (TEI XML, GML, ODF, TXT).

It is now common practice for funding agencies and host institutions to insist that projects create detailed data management plans. The Canadian Tri-Agency Statement of Principles on Digital Data Management, for example, specifies that "data should be collected and stored throughout the research project using software and formats that ensure secure storage, and enable preservation of and access to the data well beyond the duration of the research project" [Government of Canada 2016]. In practice, this means using open, well-documented standards and file formats (i.e. files that can be read, opened, and processed without the need of special software or tools) instead of proprietary formats.

<div style="text-align:right">22</div>

### 5.1.2 Data is subject to version control (Subversion, Git).

The use of version control for DH data is now widely accepted practice; by its very nature it supports LOCKSS (Lots of Copies Keep Stuff Safe) (every person checking out the repository to work on it creates a new copy), provides security against human error and data corruption, and enables sophisticated progress tracking [Holmes et al 2019]. We also believe that project history, as reflected in progressive changes to its dataset, merits preservation in itself, and version control systems support this. Jim McGrath, at our Endings Symposium, discussed in detail the importance of recognizing and surfacing contributions by students, post-doctoral researchers, and other "precarious" labour, and Jessica Otis (in this issue) also addresses this topic. Version control allows detailed tracking of contributions file-by-file, allowing for more granular visualization of responsibilities.[6]

<div style="text-align:right">23</div>

### 5.1.3 Data is continually subject to validation and diagnostic analysis.

Since all products depend entirely on the reliability of the project's dataset, it is essential to establish and maintain the accuracy of the data. For XML datasets, all files should be validated against the appropriate schemas; for other types of data, routines should be built into processing that confirm that the project's data conforms to the project's determined syntax and convention.[7] In addition to confirming that individual files validate against the project's schema, the entire dataset should also be checked using a set of what we have previously called "diagnostics" [Holmes & Takeda 2019b]. As we argued, projects contain multiple internal and external relationships — to project-defined entities, other pages within the project, or external vocabularies — that "need to be tested, checked, and validated, too, but it is impractical to do this using document-level schemas" [Holmes & Takeda 2019b, i101–2]. Taken together, standard document validation and diagnostics are necessary for confirming that a project is consistent, coherent, and complete.

24

## 5.2 Documentation

Within the digital humanities (and many other fields), the literature of best practices is suffused with exhortations to document more, document better, and document everything. The reality for grant-funded projects, however, is that documentation is frequently the last task on the agenda and therefore likely to be curtailed or neglected when funding runs out or deadlines approach [Morgan 2021]. The Endings survey found that "60% [of projects] claimed to have a clearly documented data model, but 90% of those that had documentation considered it to be partial or inadequate, so it appears that a project's data model is well documented in only about 50% of cases" [Arneil, Holmes & Newton 2019]. We are no different in our exhortations to document thoroughly, but we do take a strong position regarding the availability of documentation.

25

### 5.2.1 Data models, including field names, descriptions, and controlled values, should be clearly described in a static document that is maintained with the data and forms part of the products.

In most of our larger projects, such as *The Map of Early Modern London*, training for research assistants and other contributors forms a core part of the project's mission, and we have learned to build this kind of praxis documentation directly into the published digital edition itself. This has two significant advantages over keeping documentation segregated and private. First, since the documentation is published with the edition, it will always travel with the published product, increasing the likelihood that future researchers will be able to make sense of our materials and adopt or repurpose them easily. Second, the visibility of our documentation helps to keep us honest. Documentation is less prone to be aspirational and more likely to reflect the reality of our praxis if users of our editions can read it alongside the editions themselves, and point out any failures or discrepancies.

26

### 5.2.2 All rights and intellectual property issues should be clearly documented. Where possible the Data and Products should be released under open licenses (Creative Commons, GNU, BSD, MPL).

As suggested above, a project's best chance of long-term survival lies in making it as amenable as possible to copying and repurposing. While this article is mainly concerned with overcoming the technical barriers to digital longevity, it must also be noted that any materials or data accompanied by any sort of restrictive covenant or license will inevitably be less appealing to a future researcher considering making use of it. In the same way, the LOCKSS preservation strategy requires the collaboration of multiple institutions willing to host project products over the long term, and this is much less likely to happen if licensing issues make it more difficult to deliver the materials to the institution's end-users in a convenient way. Therefore we strongly recommend that projects, where possible, make all their materials available under generous Creative Commons licenses, open-source all their code, and avoid depending on any proprietary data, programming, libraries, or other resources. For example, it is far better to try to acquire your own copy of a historical text and scan the pages yourself than to build a digital edition around page-images supplied under license by a corporation or other rights-holder who may restrict access to them.[8]

27

Even worse than restrictive licensing is lack of clarity about license terms. If an institution or a researcher cannot discover the terms under which content has been released, they will never be able to determine whether they can or should archive or make use of it.

28

## 5.3 Processing

There is no specific way to create an Endings-compliant site; different projects demand different technologies and there is no single language, framework, or tool that will work for every case. So the principles below are necessarily technology-agnostic and, while our own personal preferences may become apparent from the examples, we do not advocate for one "stack" over another. But there is certainly no shortage of processing pipelines for creating static sites. The uptake of the "JAMstack" (Javascript, API, and Markup) — a movement that grew out of the popularity of Node.js and NPM and is now closely allied with static asset providers like GitHub, Netlify, and Amazon S3 — has lead to both the popularization and proliferation of static site generators, such as Next.js, Hugo, and Jekyll.[9] Within the digital humanities, proponents of "minimal computing" have mobilized these tools to create frameworks that significantly minimize the barriers to access in creating static digital editions and digital exhibits.[10]

But just because a pipeline produces static HTML output doesn't mean that the processing produces a site that is archivable and sustainable in the long term. The Endings team was made painfully aware of this in the creation of our own Endings website, which was built initially using GitHub Pages and a pre-built Jekyll template. Though completely static, the site violated many core Endings principles: it was comprised of malformed, invalid HTML that relied on Bootstrap 4 and a set of JQuery-based JavaScript to handle various UI components. The site has since been developed into valid XHTML5 with all JQuery dependencies replaced by CSS3 and all updates handled by XSL transformations managed by an Ant build script. The problem with the old Endings site was not that it was built with Jekyll and that, had we just used Ant and XSL in the first place, all of our problems would have been solved; rather, the main issue was that the processing was meant only to create a static site, and not one that was coherent, consistent, complete, and free of technical debt.

We understand processing, then, not in terms of a particular pipeline, tool, or programming language, but rather as the set of steps necessary for creating robust, sustainable, and archivable resources. While we have created some tools — like our diagnostics toolsets[11] as well as staticSearch, which is discussed in detail in Section 6 — as part of the Endings project, these toolsets are neither necessary nor sufficient for an Endings "processing" pipeline.

### 5.3.1 Relentless validation: all processing includes validation/linting of all inputs and outputs and all validation errors should exit the process and prevent further execution until the errors are resolved.

Our aspiration is never to release a flawed edition of any project, so validation of all inputs and products is a core requirement in the project build process. TEI XML files are always accompanied by a schema, and validation of those files is always the first step in the build; encoders should not normally commit invalid files to the repository, but this does happen from time to time. In the same way, the various other versions of XML that may be generated as part of the build process should also be validated to ensure they are functional and therefore useful to other projects. Naturally, output HTML also requires validation, and the W3C's validator, which is available as a Java JAR file (vnu.jar), is simple to use for this purpose. We use CSS code not only in HTML output but also within TEI XML as a convenient formal language to describe the layout and appearance of primary source texts; this CSS can be extracted and validated using the same W3C validator, ensuring that our descriptions are logical and processable by any CSS processor.

Standard validation can catch many syntactic and stylistic errors in individual documents, but they do not necessarily confirm the collection's coherence and completeness as a whole. As we have argued before [Holmes & Takeda 2019b], validation should be accompanied by a set of project diagnostics, which can, among other things, confirm referential integrity between documents, check for potential duplication of data (people, places, or bibliographic citations), and trace a project's progress across time.

### 5.3.2 Continuous integration: any change to the source data requires an entire rebuild of the site (triggered automatically where possible). Processing prioritizes validity and maintainability over speed and efficiency.

At its most basic, the processing for an Endings-compliant site could simply be a single individual who lovingly encodes an HTML document by hand and, after validating it against the W3C's HTML validator, uploads it to their server. Most projects, however, require a more robust technical pipeline that can rearrange, transform, and aggregate their source materials into a fully functioning web application. While these builds should be able to be run locally, the project team seldom requires a local version as the project building is managed by a Continuous Integration and Continuous Deployment server (CI/CD). Our Jenkins server polls each repository and whenever it detects any change, it re-runs the entire build process, deleting all of the old artifacts and creating them anew.

Compared to server-side systems that prize rapid delivery of products in milliseconds, our build processes can seem painfully slow and massively inefficient: changing a single character in a single file initiates an entire re-build of the site, which, depending on the size of the project, can take anywhere from a few minutes to two hours. Much of that time is spent not on building the HTML products that comprise the final site, but on relentlessly validating any and all inputs and outputs. However, this processing is mechanical and generally requires no oversight at all.

One might question why a single change to a single document requires an entire project rebuild. But any given change may have significant repercussions throughout a document collection. A change to someone's biography may affect every page which mentions that person; a change in a gazetteer may propagate through dozens of maps. In practice, it is often very complicated to pin down exactly which components of a site must be rebuilt following a change, and it is far more straightforward just to rebuild everything. This also ensures that site-wide edition/version information (which in the case of our sites usually includes the source code repository revision number from which the site was built) is consistent in every page footer.

### 5.3.3 Code is contingent: while code is not expected to have significant longevity, wherever possible, all code should follow Endings principles for data and products.

Much like the source data, all processing code should be subject to version control and should ideally reside in the same repository as the source data, which ensures that the data and the code are always in sync and thus makes rebuilding a project at a particular version trivial. This also means that all dependencies, including external JAR files and binaries, should be stored in the repository when the terms of the software permit (open-source software here is, of course, preferred). While it is a good idea to build routines into your processing that check for updates, upgrading a dependency should always be the developer's responsibility and not something performed silently in the background by a package manager; the repository should always contain the last version of the dependency that works with the rest of the code, such that the processing can work even when that version, the organization, or the package registry disappears.

However, while data and products are designed to survive, processing code is necessarily impermanent; while we strive to create the appropriate conditions for maintaining reproducibility, there is no guarantee that code will work in a year, let alone fifty. This is a luxury afforded by static sites: while server-side infrastructures require a great deal of consideration and investment to ensure that the processing can be maintained in future, it makes no difference how a static site is created as they are, by definition, untethered from the processing that created them.

## 5.4 Products

Principles for products are at the heart of the Endings approach, since products are the only components of a project that we really hope and expect will survive. This section of the Principles outlines the technological choices, structure, and organization we recommend for digital edition projects aiming for longevity.

### 5.4.1 No dependence on server-side software: build a static website with no databases, no PHP, no Python.

"Lots of Copies Keep Stuff Safe," says the conventional wisdom, and Stanford has an entire digital preservation program (https://www.lockss.org/) named for this truism; certainly, the more copies of something that exist, the more likely it is to survive over the long term, especially if those copies are widely distributed geographically (see [Cayless 2010]). In the world of online digital resources, though, it doesn't really matter how many archival copies of your TEI XML source encoding are distributed across the world; if the one hosting server running the database backend and WordPress front-end for your digital edition goes down, your project has essentially disappeared.

We have therefore focused on building digital editions which can run on any web server, anywhere, without any specific dependencies. If spinning up a new site requires nothing more than copying a collection of files to a server and circulating the URL, there is a far greater chance that *functional* copies of the *products* of your work will survive in a usable form. Every server-side dependency is a barrier to replication and therefore to survival.

**5.4.2 No boutique or fashionable technologies: use only standards with support across all platforms, whose long-term viability is assured. Our choices are HTML5, JavaScript, and CSS.**

**5.4.3 No dependence on external libraries or services: no JQuery, no AngularJS, no Bootstrap, no Google Search.**

The World Wide Web — the front-end of the modern Internet — is perhaps the most successful and prolific invention in the history of human communication. Its current scale dwarfs the entire prior history of text, and it continues to expand at an astonishing rate. Underlying its functionality are three core languages: the markup language HTML, the style language CSS, and the scripting language JavaScript (ECMAScript), all of which are well-managed by standards bodies. This trio of technologies underlies more than the web, of course; cell phone applications, EPUB documents, and many other forms of communication are also built on the base technologies of the Web, all of which are relatively simple and easy even for beginners to understand and learn.

However, the world of content-creators who build Web resources is not so simple; rarely do developers sit down and code HTML documents and write plain CSS. Instead, they tend to import large packaged libraries of existing code to generate the end-user Web pages we consume. Large coding frameworks for creating web-based resources come and go at a remarkable speed. At the time of writing, Angular, React, Vue, Express.js, Meteor, and Node are all popular JavaScript frameworks, while JQuery, Dojo, MooTools, and others have fallen out of favour; by the time you read this, the situation will certainly have changed again. The same is true of database-dependent content-creation and delivery tools such as WordPress, Ruby on Rails, Drupal, Joomla, and others, as well as all of the tempting "free" services that have enabled DH practitioners to create intriguing mash-ups and to decorate their work with data from other sources. All of these efforts promise rapid site development at the cost of long-term maintenance issues. As Jim Nielsen (2021) and others have pointed out, this creates an ecosystem in which "all web languages — HTML, CSS, and JS — are compile targets." Programmers, in other words, no longer need to code directly in the target languages and instead become specialists in a few fashionable frameworks, chasing the changing fashions year by year.

And yet, the trifecta of HTML, CSS, and JavaScript is remarkably powerful. We know that because ultimately, all of those other languages, frameworks, and tools — from MySQL+PHP to PostGreSQL+Python to RethinkDB+Node.js — basically do one thing: they produce HTML/CSS/JavaScript Web pages, and those Web pages do all the things we need them to do. And while each of those technologies or frameworks or back-end services will eventually stop working, it is extremely unlikely that Web pages themselves will cease to function. One of the most remarkable things about HTML, CSS, and JavaScript is that over 20+ years of development, they have retained impressive levels of backward compatibility. The first Web pages ever created still work perfectly (http://info.cern.ch/hypertext/WWW/TheProject.html) and, as noted above, the first projects we created at HCMC in the 1990s and early 2000s are also still perfectly functional. Given the astonishing quantity of resources built and delivered through HTML, CSS, and JavaScript, there is a strong chance that they will continue to function over the long term; and when they do, perhaps, alter in ways that make older forms of text less usable, there will be readily-available migration pathways and tools that can bring along everything that survives and is worth preserving. The same will not be true of this year's favourite JavaScript framework or last year's most popular content management system.

In the same way, if maintenance is ever required, it is very likely that there will be programmers who are able to read and understand the three core languages. It is not so certain that programmers capable of debugging and fixing a ReactJS application or a PHP script will be so common. Building your products from the three core technologies substantially increases their chances of survival; dependence on back-end or external services is a temporary solution for short-term projects.

**5.4.4 No query strings: every entity in the site has a unique page with a simple URL that will function on any domain or IP address.**

As noted above, older sites which queried back-end databases to construct their "pages" tended to have page URLs whose distinctive factors lay in the query string rather than in the base location. For example, *The Robert Graves Diary* project formerly required the following URL to retrieve the diary entry for 22 February 1935 (line-wrapped for easier reading):

```
http://graves.uvic.ca/graves/site/xbrowse.xq?
```

```
collection=%2Fdb%2Fgraves&type=diaryentry&
query_stored=false&action=browse&
search_text=-1&day=22&month=02&year=1935
```

This concoction presented obvious difficulties for the Internet Archive crawler, so none of the diary entries were actually archived in the Wayback Machine from the original site. Contrast this with the current equivalent:

```
https://graves.uvic.ca/diary_1935-02-22.html
```

which itself is linked from a sitemap page to ensure its retrieval by standard crawlers (see [Holmes 2017b]). Such URLs are not only cleaner and simpler for automated tools to process; they can also be manipulated by humans who happen to know exactly what date (in this case) they are looking for, removing the additional step of re-engaging with form controls or other GUI components when browsing the project. When rewriting *Le Mariage sous L'Ancien Régime* as a static Endings site, we were able to take advantage of our institution's involvement in the Archive.org project to deploy the Internet Archive's Heritrix crawler repeatedly on a pared-down version of the digital edition, inspecting the harvested results and tuning the site organization, linking, and filenaming until we were able to ensure that everything important was being successfully crawled and archived.

In addition to ensuring that every URL is simple and meaningful, we also believe that every entity (document, prosopography entry, bibliography entry, collaborator biography) should have its own unique page with its own URL. This provides maximum flexibility in terms of linking, extraction of individual pages, and re-use by future researchers (see below). Many metadata schemes (such as RDF) rely on URLs as unique identifiers for categories, concepts, and entities; a scholarly publication can embody this approach, and support Linked Open Data applications more effectively, by segmenting entities at the page level in the site structure. 47

### 5.4.5 Inclusion of data: every site should include a documented copy of the source data, so that users of the site can repurpose the work easily.

Hugh Cayless (2010) points out that many ancient texts have come down to us only because of a long tradition of copying, reuse, quotation, and repurposing, and suggests that "we have returned to a situation somewhat like the one that existed in the ancient world and furthermore that perhaps some of the processes that governed the survival of ancient works might pertain to digital media. As in ancient times, a work released into the electronic environment may be copied, quoted, reused or resold without the originator's having much control over what happens to it" [Cayless 2010, 147]. We subscribe to the view that such copying and reuse is not only inevitable but profoundly desirable; although we hope and trust that our project editions will survive intact and remain functional over the long term, it is just as likely (perhaps far more likely) that fragments — either a small subset of documents or even a single page — of the project will be taken, repurposed, and reshaped by future scholars. 48

Therefore in projects such as *The Map of Early Modern London*, we create a variety of different versions of our XML, each intended for particular types of re-use ([Holmes 2017a]). One version is a "standardized" TEI in which all the most unusual or edge-case encoding features have been converted to more commonly-used encodings. Another is a "standalone" TEI file into which all linked records from prosopographical resources, gazetteers, and other centralized data sources from the collection have been copied into the document, creating a single archive which can be taken and reused without any dependency on the constellation of surrounding linked resources. Our intention is to make it as easy as possible for any future scholar or student with an interest in a particular text to download an XML version of it which is ideally suited to their needs and can form the basis for their own project or research without special preparation. 49

### 5.4.6 Massive redundancy: every page contains all the components it needs, so that it will function without the rest of the site if necessary, even though this means duplicating information across the site.

Just as we offer repurposable XML versions of our documents, we try to provide HTML pages which can also be pulled out of the project and reused easily. While TEI XML is a widely-used and trusted archival format for digital scholarly work, compared with HTML, its audience is tiny. As we have noted elsewhere [Holmes 2017b], the many billions of HTML pages already created constitute many times the number of printed books produced in the entire history of humanity, and the variety of purposes already served by HTML (not only websites but mobile applications, archival storage, and others) will ensure that if anything from the current era of digital communication survives, it will be HTML. So our HTML is more likely to be reused than our XML in the long term.

51

Since we want to encourage this, it makes sense to provide documents in a form easily excised from the collection. Documents should, as far as is practical, stand alone, and therefore we usually generate our HTML outputs from the standalone versions of our XML source. Rather than having a document link to thirty external prosopography entries that live in other documents on the site, why not simply incorporate copies of all those entries inside the document itself? Text is typically small and compressible compared with some other digital resources. If we make a point of replicating all required linked records such as bibliographic citations, toponymic variants, and personography entries inside each individual document, any user can take a single HTML page and use it elsewhere without having to resolve a large number of pointers and retrieve a set of external resources to make the document functional. This of course means that the same individual personography entries may be replicated hundreds of times across a collection. This seems wasteful; but as we have said previously [Holmes & Takeda 2019a], "We don't care."

### 5.4.7 Graceful failure: every page should still function effectively even in the absence of JavaScript or CSS support.

Graceful failure (or degradation) of internet services has been discussed and recommended since the 1990s (see for example [Jakob Nielsen 1999]). However, while widely acknowledged, it is rarely implemented in a practical way. Many modern websites depend entirely on JavaScript and CSS and cannot function as plain HTML (imagine Google Docs or Office 365 without scripting), which makes it impractical for ordinary users to turn off JavaScript in their web browsers; consequently, no-one ever does, and graceful degradation is never needed or tested in the real world.

However, static websites lend themselves rather well to graceful degradation. If a page already contains all the secondary or supplementary information it requires in the form of biographies, bibliographical data, and so on, as we recommend above, then it can easily function effectively in the form of plain HTML, just like the earliest web pages of the 1990s. Through progressive enhancement, JavaScript may turn links to footnotes into buttons that generate convenient popups, but the underlying page structure can be plain and functional even in the absence of CSS and JavaScript. This makes "relentless" validation and link-checking trivial to implement during the processing stage.

Another advantage of this approach is that it conforms with the relatively new (at the time of writing) Content Security Policy header settings which are rapidly gaining traction. For example, the current default implementation of CSP settings will block the use of (very common) inline JavaScript or CSS[12] unless CSP explicitly includes "unsafe-inline" settings. It is very difficult to predict how CSP and other security headers may evolve over the next few years, but it is not unlikely that standard and preferred constraints on the majority of web servers may block many features which are currently widely used, so progressive enhancement is more important than ever.

The principles above are tempered by the following concessions:

55

### 5.4.8 Once a fully-working static site is achieved, it may be enhanced by the use of other services such as a server-side indexing tool (Solr, eXist) to support searching and similar functionality.

Although it is perfectly practical to create an entirely static client-side JavaScript search engine for digital edition sites (see below, where we discuss the staticSearch engine we have created and use on all our static sites), it may be more convenient or practical to take advantage of search functionality available from a third-party provider such as Google, or an institutional host running a central search engine such as Solr. Indeed, it might be argued that search functionality is not a fundamental component of a digital edition, but rather a feature of the broader context of the Internet itself. However, if you take this view, and create an external dependency, it is vital to realize that the dependency is likely to fail at some point, and to consider that this feature of your digital edition is nonpermanent. External systems alter their APIs as well as their terms

and conditions frequently, and if no-one is around to maintain the articulation of such dependencies, the functionality will not persist. This may be an acceptable compromise; for example, in the case of a small site with limited programming resources, it may be reasonable to assume that some kind of generalized search engine such as Bing or Google will always be available, and that such tools will automatically index a site and provide sophisticated users with methods of searching within that specific site, and in such a case, a dedicated search engine is not required. On the other hand, most digital editions have rich metadata that lends itself well to customized search pages with search filters based on date, document type, language, and many other idiosyncratic features of their specific dataset which will never be supported by a generic search engine.

### 5.4.9 The use of an external library may be necessary to support a specific function which is too complex to be coded locally (such as mapping or cryptography). Any such libraries must be open-source and widely-used, and must not themselves have dependencies.

While we are strongly in favour of the injunction in 5.4.2, "No boutique or fashionable technologies," and recommend the avoidance of large general-purpose web development platforms such as React or Angular, there are cases where using an external library is the only sensible option. One example is cryptography: as Jakobsen and Orlandi note, "well-studied, provably secure encryption schemes that achieve strong definitions of security (e.g., authenticated-encryption) are to be preferred to home-brewed encryption schemes" [Jakobsen & Orlandi 2016, 113]. Similarly, while it is possible to write your own mapping tools, the cost in time and money to do so for a single project is most likely prohibitive; it is much more practical to import and use an existing library such as Leaflet or OpenLayers. The important factor when integrating external libraries into a codebase is to choose them wisely. A relatively small library which is designed to do one specific job (such as mapping or cryptography) very well, and itself has no external dependencies, is a much better prospect than a general-purpose collection of functions and APIs trying to do a thousand things and pulling in hundreds of secondary dependencies to support them. 57

Similarly, for some features of a site, it may be essential to depend on an external service of some kind. Mapping is another obvious example here; very few projects are going to be in a position to run their own map tile servers, so they will rely on publicly-available services such as Open Street Maps or Bing Maps. These services will eventually cease operation, change their APIs, or change their terms of use, and the maps will stop working, but as long as they are not crucial to the main purpose of the site, this is not a disaster. The important thing is to build this expectation into the project plan. 58

Above all, as in the case of external searching or indexing tools, incorporating a dependency incurs a technical debt, and may result in a failure of functionality in the future. For this reason, it is important to ensure that such external dependencies are not providing functionality which is completely crucial to your digital edition. You can also mitigate the possibility of such failures by ensuring that (for example) your site has rich and exhaustive pre-compiled index pages so that users can find their way to desired documents without using a search engine, and that key components such as the names of people and places help to provide a network of linking throughout the edition, offering the user many paths through the data. 59

## 5.5 Release Management

Without good release management, a project can never end gracefully; it can only falter and die. 60

These principles apply to release management: 61

### 5.5.1 Releases should be periodical and carefully planned. The "rolling release" model should be avoided.

In the world of traditional print scholarship, a publication is a coherent singular object, released on a particular day in a particular place, and provided with a convenient edition number which enabled scholars to cite it without ambiguity.[13] However, many digital edition projects have adopted a "rolling release" approach modelled on the predominant approach to software publication, where corrections and new features are steadily made to a mutable product. Typos and errors on a single page are quickly fixed and the site can be endlessly tweaked as new problems are inevitably discovered. 62

This "rolling release" model, however, makes it difficult to maintain consistency, coherence, and completeness as the site is in a perpetual state of flux; pages are constantly subject to change, and thus the project as a whole is always shifting. Not only does this make stable citation problematic, but it makes archiving difficult. Most large projects go through moribund phases in which work largely stops, and will most likely end in a similar way, and so it is essential that whatever is the current released state (edition) of a project constitutes an acceptable version for the long term. The edition model of digital 63

publication ensures the stability and coherence of the project; much like a print publication, each edition of a digital publication should be planned carefully, with deadlines and milestones governing its release.[14]

### 5.5.2 A release should only be made when the entire product set is coherent, consistent, and complete (passing all validation and diagnostic tests).

The irreversible nature of print publication means that an edition is often scrutinized and carefully proofed prior to publication. In the "rolling release" model, since digital material is much easier to correct and changes can be applied soon after they are found, it is far less common for this level of diligent inspection to be applied to digital editions. Since each release is organized around a set of milestones and goals, much of this scrutiny can be automated (see 5.3.2). In [Holmes & Takeda 2019b], we describe three distinct levels of diagnostic checks which can be incorporated into a project build process to provide mechanical proof that no links are broken, no content is missing, and all planned content and features are complete. This approach, combined with the detailed proofreading we would expect to apply to any scholarly publication prior to release, not only minimizes flaws in the released edition, but also ensures that each edition is archivable.[15]

### 5.5.3 Like editions of print works, each release of a web resource should be clearly identified by its build date and some kind of version number.

Just like print editions, digital resources should carry a clear edition number on every page, which applies to every part of the resource. Our normal practice is to include this information in the page footer. At the time of writing, for example, the current version of *The Map of Early Modern London* contains this information in its footer:

> MoEML v.6.5, svn rev. 17540 2020-09-15 12:35:49 -0700 (Tue, 15 Sep 2020).

This shows not only the specific edition number (6.5), but also the Subversion repository revision from which it was built, along with the exact date and time of the build.[16]

### 5.5.4 Web resources should include detailed instructions for citation, so that end-users can unambiguously cite a specific page from a specific edition.

Citation patterns for web-based resources are still subject to some confusion and change, so we recommend that projects make things easier for other scholars by providing copy/pastable citation blocks for each individual page, accessible from a link on the page. The example in Figure 3 comes from the *Mapping Keats's Progress* project.
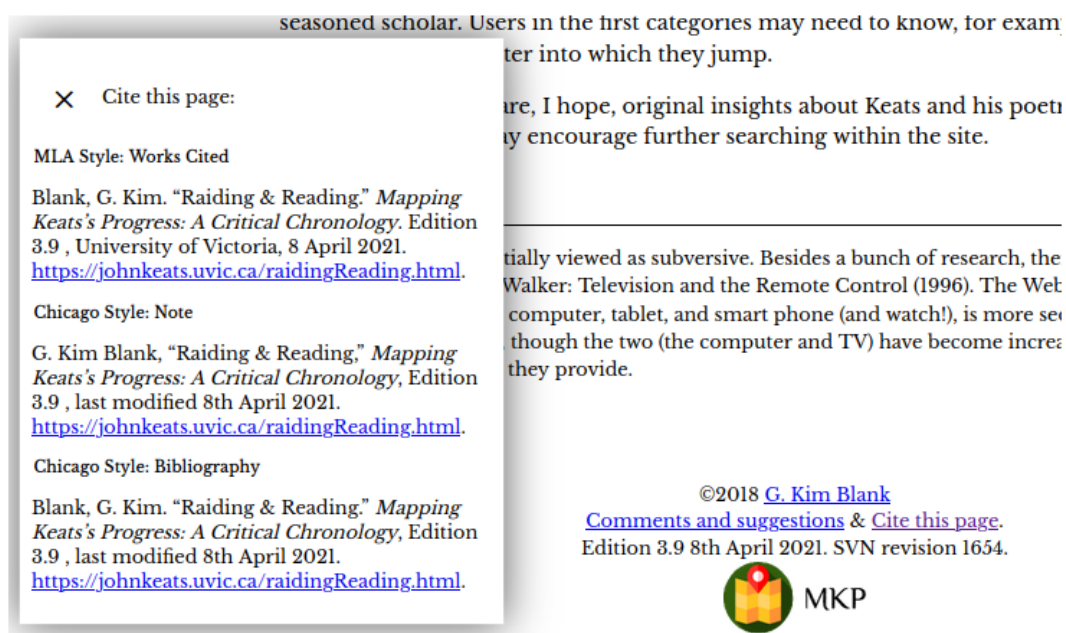
**Figure 3.** Citation popup in *Mapping Keats's Progress*.

### 5.5.5 URLs for individual resources within a digital publication should persist across editions. Any moved, retired, or deleted resources no longer available at a previously accessible URL should be redirected

**appropriately.**

The same resource in a digital edition should appear at the same URL in subsequent editions. Where a resource is removed, retired, or merged with another resource — when, for example, a person in a personography proves to be the same individual as another record, or is proved not to have existed at all — then the original URL should redirect to the replacement URL, or to an explanation of why the original resource is not available. This is essential for maintaining "cohesion and integrity in the scholarly record" (Coble and Karlin in this issue).[17]

## 6. The Lingering Problem: Search

With these Principles in place, we were able to convert the dormant projects, such as *The Robert Graves Diary*, and the on-going projects, like *The Map of Early Modern London*, from unwieldy eXist applications into a large bundle of static HTML, CSS, and JavaScript. Our intention was that these static incarnations look and function identically to the previous live application and, as these sites were steadily released and tested, we were happy to find that this process was not only feasible, but significantly improved the projects. Our staticization of *Graves*, for instance, surfaced various encoding errors that had gone unnoticed for many years, which we could now easily diagnose and fix. And in moving from a dynamic application to the static site, we had the opportunity to re-write the codebase from scratch, removing the layers of obsolete code that had sedimented within the codebase over its long history of updates, maintenance, and modernization of the CSS and JavaScript. Projects in active development benefitted similarly; these re-writes provided a good opportunity to scrutinize the project, lopping off irrelevant or deprecated code. Thanks to our Jenkins Continuous Integration server, following any change to the source data or the processing, we no longer needed to click anxiously around the site to confirm that we hadn't broken the anything: any encoder could simply commit, wait a few minutes, and then, so long as the Jenkins server didn't send its "BUILD FAILED" message, proof their changes in the context of the site.

But these sites all lacked a significant and crucial part of their functionality: search. Most digital editions require some form of text-based searching and, in many cases, the ability to search across a large and disparate document collection and aggregate documents based on multiple metrics is the project's raison d'être. Search was, in other words, non-negotiable — we could not sacrifice the robust search mechanisms on the existing site. We could quietly ignore search for the time being in some projects, like *The Map Of Early Modern London*, where search was peripheral to the main functionality, but *The Robert Graves Diary* forced us to come up with a real solution: the homepage for the project was essentially one large, faceted search page, which we were able to replicate perfectly, but since search itself was handled entirely by eXist, the static version was rendered completely inert.

**Figure 4.** The original home page of *The Robert Graves Diary*.

We had always assumed that the kinds of search our projects required — complex faceting and filters, wildcards, and exact phrase queries split across multiple, collection specific search pages — was necessarily the stuff of servers and could not be made static. While some client-side search solutions were available, they were ill-equipped to handle the thousands of

documents, millions of words, and complex metadata structures that comprise a standard digital edition.[18] Outsourcing indexing to an external service, like Elasticsearch, was out of the question as it would violate the core Endings principles.

Our intermediate solution was to stick with what we knew: package the static sites into a XAR bundle and spin up a simple eXist instance whose only job would be to index, query, and retrieve the search results from the static HTML pages. Of course, this worked perfectly, and we released the first version of a number of our static sites following this approach. But it was a disappointing compromise. We had gone through the trouble of re-writing these sites in their entirety to remove the fragility inherent in their dependency on eXist, and had celebrated our success in building completely static, serverless websites, and now here we were, returning to eXist, hat in hand.

<span style="float:right">71</span>

If we were to return to eXist, we reckoned, then we should at least try to improve the situation; in the spirit of LOCKSS and following our principle for "Graceful Failure" (5.4.7), we decided to take a decidedly maximalist approach. Using *Graves* as our case-study, we supplemented eXist with three distinct approaches, which all had their own limitations with respect to functionality, archiving, and sustainability, but could provide something of a safety net for the project. We discuss these approach in greater detail in [Holmes & Takeda 2018], but, in brief, those three approaches were: 1) a Google search widget, which would not offer anything beyond what Google already provides, but at least gave users a shortcut for a site-specific search; 2) an experimental interface for querying a Solr index provided by the UVic Library (the eventual archival home for all of these projects), which could provide many of the faceting and filtering structures required by the project; and 3) a "standalone search," a Javascript-only client-side text search interface that would stem user input and match it against an pre-built inverted index generated during the build process. In this process, a list of distinct words was created for each document; the words were then run through the Python implementation of the Porter Stemmer, and then documents were grouped by stem, so that JavaScript could return a simple list of the documents in which that stem appeared. This solution, as we noted then, worked well for small document collections, but lacked the necessary features — keyword-in-context, search faceting, and filtering — and thus should be considered the "ultimate fallback when all else fails" [Holmes & Takeda 2018, 59].

<span style="float:right">72</span>

Over the last three years, we have continued to improve and expand what we then called "standalone search" into "staticSearch": a fully-featured and sustainable search engine for static websites that has no server side dependencies. Now implemented by nearly a dozen projects, staticSearch has not only allowed us to replicate server-side search functionality, but has significantly expanded what these searches can do. So to revise our answer to the question from our 2018 paper, "Why do I need four search engines?": You don't.

<span style="float:right">73</span>

## 6.1 How staticSearch works

Simple client-side search engines such as Lunr.js work by generating a large index file, which is then downloaded to the client, and queried by JavaScript in the search. Obviously, the larger a document collection, the larger the index, and any site consisting of more than a few dozen pages will rapidly outgrow this approach. The key insight which enabled us to create staticSearch was the idea that the search page need only download those components of the index it actually requires to respond to a specific query. Instead of a single large index, we create thousands of individual JSON files, one for each stemmed term, and the text search component of the search page simply stems all the search terms and retrieves only the specific JSON files for each of those terms. This enables rapid query response even for large sites.

<span style="float:right">74</span>

Of course, text search alone is not enough; search facets such as topic, document type, date range, and other metadata-based filters are a fundamental requirement for searching digital editions and other rich document collections. One by one, we have added support for a range of these features, which can be configured by the addition of HTML `<meta>` elements with specific patterns to site pages. The staticSearch indexing process works its way through the HTML pages of a site indexing the text but also searching for these `<meta>` elements, building a collection of distinct JSON files which the search page can retrieve as needed to respond to queries and filter results.

<span style="float:right">75</span>

**Figure 5.** One of several specialized search pages built using staticSearch for the *Digital Victorian Periodical Poetry* project.

[Holmes & Takeda 2020] provides a more detailed explanation of how staticSearch works, and the full documentation for the current release is available on the Project Endings website.[19]

# 7. Archiving

The technical principles that we have outlined so far demonstrate how we have created resources that can be archived, or that, at least, lack the technical barriers that have made digital preservation difficult (or impossible) in the past. However, there are still a number of issues and challenges that the Endings project has faced in preparing to archive these projects within institutional repositories and, in particular, in preserving the form and function of these projects such that they can continue to be useful as scholarly resources.

While librarians and archivists have long recognized the importance of secure archival storage, digital archiving initiatives have largely focused on data storage and management (see [Molloy 2011] or [InterPARES 2013] for examples). Back in 2002, for instance, Gelaw et al. noted that "It is now common knowledge that digital information is fragile in ways that differ from traditional technologies, such as paper or microfilm. The fact that information is increasingly stored in digital form,[sic] has led to an accelerated search for effective methods of managing electronic information resources." However, the very title of that article, "A Metadata Approach to Preservation of Digital Resources," underscores that, in many cases, archiving is primarily a matter of metadata; as they note, the "creation of preservation metadata for electronic files and digital collections are [sic] among the most important steps." This approach treats understanding digital resources as discrete blobs to be stored in large repositories with their accessibility and findability contingent on the provision of effective descriptive metadata. As Goddard and Seeman write, "in order to ensure longevity, the library will make multiple copies of our digital collections, aggregate them into Archival Information Packages (AIPs), and distribute these archival copies in offsite locations" [Goddard & Seeman 2020, 40].

For end-users not able to access archiving services from libraries or similar institutions, there are also archiving services available on the Web, such as Zenodo (offered by CERN, https://zenodo.org/), Preprints.org, or HAL (https://hal.archives-ouvertes.fr/). All of these, however, follow a similar model: a single binary object such as an article is uploaded and associated with metadata. Datasets may also be attached, but these are viewed as objects to download. This kind of archiving, while it may be a useful backup strategy (and we can never have enough of those), will not make a digital edition project (i.e. a complex website) available in the way a user would want to access it: as a functioning website. The Text Encoding Initiative, for example, archives each version of its standards package, consisting of schemas, exemplars, and other documentation, on Zenodo, but the end user can only download a zipped package of materials; they cannot browse and read the TEI Guidelines itself, as they can on the TEI's own website. As we have suggested above, if a digital edition is to remain relevant, to continue to be read, cited, and hold its place in scholarly dialogue, it must be fully functional, available, and usable; after all, as Morreale reminds us, "Humanities scholars have been taught that what they produce should be, first and foremost, consumed and understood by other scholars. Once this is accomplished, they want those who have received their work to reuse or build upon it, so that their contributions become part of a larger conversation, one that may stretch out for years, decades, or even centuries" [Morreale 2019, 4]. However, a shelved DVD copy with metadata in a catalogue or a zip archive in a distributed repository does not facilitate the kinds of experience that are at the heart of a digital project; only a functional website does. Sacchi (2015), based on a now-inaccessible technical report from the National Archives of Australia, offers a useful distinction between "stable artefacts" — resources preserved as blobs — and "performance" [Sacchi 2015, 51ff]. "Performance" here means the direct, live access to digital objects, which, like any live performance, is a unique experience bound to a specific occasion and is contingent on the availability of hardware and software components.[20] Understanding digital archives as not just "stable artefacts" but also as "performance," as Sacchi writes, "explicitly addresses the need for something concrete that needs to [be] experienced by a researcher in order [to] access the essence of a record" [Sacchi 2015, 53]. In other words, while it is essential that digital resources are archived as "stable artefacts," preserving the "essence" of a digital edition requires maintaining its functionality and allowing users to experience it as "performance."

Librarians and archivists are now beginning to focus on this important distinction. The University of Victoria Library is currently developing a "Donation policy for legacy websites," which outlines the conditions under which an individual or a project group may transfer an entire website to the library for long-term storage, with the expectation that it will actually remain accessible as a website on the Internet for public access. Offering such a service is not simple, though. A library presented with a Tamagotchi-style web application, with dependencies on back-end databases and server-side scripting, will (and should, quite rightly) reject the "donation" as it would, as we have argued, amount to the passing on of technical debt and the obligation for incessant maintenance. But the static site model we have proposed and implemented is not subject to these considerations. Assuming any functional webserver is available, a static site can be served for the foreseeable future with no maintenance burden whatsoever; only bandwidth and disk storage are required, just as is the case for binary-blob digital assets in the traditional archive model. With a static site approach, any given edition of a digital collection could be available in multiple "locations" (i.e. at multiple URLs), hosted by different institutions, providing resilient and reliable access to the resource even in the face of occasional failures or closures at specific institutions.

Such a scenario presents its own challenges, though. If a site was originally offered at a custom domain (www.ourfabulousproject.net), should the archiving institution agree to take on the ongoing cost of preserving that domain? If there are multiple incarnations of a site, how is it to be cited, and how is a reader to know that they are viewing the "same" site as that cited in a publication, albeit perhaps on a different URL? This is one reason that the edition management and release model proposed in section 5.5 is so crucial. Each coherent, consistent, and complete edition of a digital project may be archived and served from many locations, so it must be reliably and obviously consistent and show clear dating and versioning information, just like an edition of a printed book.

## 8. Conclusion

When Project Endings began, we really had no idea how practical it would be to build interactive digital editions without back-end services, but our initial work was very encouraging, and in fact it proved relatively straightforward to rewrite even very large and complex sites such as *The Colonial Despatches* (over 7,000 TEI-encoded documents) and *The Map of Early Modern London* (over 12,000 pages) into simple, fast, responsive, and easily searchable static sites. Other sites from our growing collection have taught us to handle a variety of different types of data such as video (*Francotoile*) and image collections (*MyNDIR*), while the largest dataset, the *Landscapes of Injustice Archive*, presents a searchable collection of

over 150,000 pages representing diverse data including maps, oral histories, land title records, street directories (in Japanese and English), correspondence, and legal casefiles.

At this point, we can say with confidence that the approach we have taken is practical and effective. You *can* build digital editions this way, and it really doesn't take much more work than it would to roll out a Tamagotchi of frameworks, libraries, software, and infrastructure. Once it's done, it takes no maintenance at all, and you're free to move on to the next project or focus on the next edition.

83

We're doing it. You should do it too. You'll regret it if you don't.

84

# Appendix 1: Endings-compliant projects

This is a list of all the projects which have been constructed or reconstructed in accordance with the Endings principles outlined in this article.

85

## The original four projects

- The Map of Early Modern London
- Le Mariage sous l'Ancien Régime
- The Robert Graves Diary
- The Nxa'amxcín Database and Dictionary (currently not public)

## Projects later adopted by Endings

- The Colonial Despatches of Vancouver Island and British Columbia
- Digital Victorian Periodical Poetry (site still under development)
- Francotoile
- The Landscapes of Injustice Archive
- Mapping Keat's Progress
- Modernist Versions Project
- The Scandinavian Canadian Studies Journal (Moved in 2022 to another host institution, and no longer Endings-compliant)
- The Winnifred Eaton Archive

# Appendix 2: Endings Principles (version 2.2.1)

We divide digital projects into five primary components:

86

1. Data
2. Documentation
3. Processing
4. Products
5. Release management

## 1 Data

Data is the expression of the source information, knowledge, and expertise of our researchers. The following principles apply to data:

87

1.1 Data is stored only in formats that conform to open standards and that are amenable to processing (TEI XML, GML, ODF, TXT).

88

1.2 Data is subject to version control (Subversion, Git).

89

1.3 Data is continually subject to validation and diagnostic analysis.

90

## 2 Documentation

2.1 Data models, including field names, descriptions, and controlled values, should be clearly documented in a static document that is maintained with the data and forms part of the products.

2.2 All rights and intellectual property issues should be clearly documented. Where possible the Data and Products should be released under open licenses (Creative Commons, GNU, BSD, MPL).

## 3 Processing

Processing code is written and maintained by the project technical staff, and is also subject to version control. Processing code provides all the following functions:

3.1 Relentless validation: all processing includes validation/linting of all inputs and outputs and all validation errors should exit the process and prevent further execution until the errors are resolved.

3.2 Continuous integration: any change to the source data requires an entire rebuild of the site (triggered automatically where possible).

3.3 Code is contingent: while code is not expected to have significant longevity, wherever possible, all code should follow Endings principles for data and products.

## 4 Products

Products are the project outputs intended for end-users, typically in the form of websites or print documents. The following principles apply to products intended for the web:

4.1 No dependence on server-side software: build a static website with no databases, no PHP, no Python.

4.2 No boutique or fashionable technologies: use only standards with support across all platforms, whose long-term viability is assured. Our choices are HTML5, JavaScript, and CSS.

4.3 No dependence on external libraries or services: no JQuery, no AngularJS, no Bootstrap, no Google Search.

4.4 No query strings: every entity in the site has a unique page with a simple URL that will function on any domain or IP address.

4.5 Inclusion of data: every site should include a documented copy of the source data, so that users of the site can repurpose the work easily.

4.6 Massive redundancy: every page contains all the components it needs, so that it will function without the rest of the site if necessary, even though doing so means duplicating information across the site.

4.7 Graceful failure: every page should still function effectively even in the absence of JavaScript or CSS support.

These principles are tempered by the following concessions:

4.8 Once a fully-working static site is achieved, it may be enhanced by the use of other services such as a server-side indexing tool (Solr, eXist) to support searching and similar functionality.

4.9 The use of an external library may be necessary to support a specific function that is too complex to be coded locally (such as mapping or cryptography). Any such libraries must be open-source and widely-used, and must not themselves have dependencies.

## 5 Release management

Release management handles the public release of products. These principles apply to release management:

5.1 Releases should be periodical and carefully planned. The "rolling release" model should be avoided.

5.2 A release should only be made when the entire product set is coherent, consistent, and complete (passing all validation and diagnostic tests).

5.3 Like editions of print works, each release of a web resource should be clearly identified on every page by its build date and some kind of version number. `111`

5.4 Web resources should include detailed instructions for citation, so that end-users can unambiguously cite a specific page from a specific edition. `112`

5.5 URLs for individual resources within a digital publication should persist across editions. Any moved, retired, or deleted resources no longer available at a previously accessible URL should be redirected appropriately. `113`

## Notes

[1] https://www.exist-db.org/exist/apps/homepage/index.html.

[2] https://en.wikipedia.org/wiki/Write_once,_run_anywhere.

[3] In our Endings survey, for example, 31% of projects used a MySQL database, 15% used Drupal, and 18% used WordPress; others depended on Ruby/Rails, Django, Omeka, Plone, Cocoon, and eXist.

[4] The Appendix lists most of the static sites we have produced so far.

[5] https://en.wikipedia.org/wiki/Tamagotchi

[6] See, for example, this video of SVN commits generated using Gource from MoEML's svn repository: https://www.youtube.com/watch?v=Qor69c2Cvmc.

[7] We discuss this processing in greater detail in 5.4.

[8] There are of course contexts in which it may be impossible to make all data publicly available; one of our own Endings projects, the *Nxa'amxcín Database and Dictionary*, contains culturally-sensitive personal and linguistic data which cannot be shared outside the original language community.

[9] At the time of writing, the JAMStack website lists 326 generators across 45 programming languages.

[10] Some of the most popular of these tools are the GO:DH Minimal Computing Group's digital-edition platform "Ed" and their digital exhibit builder "Wax" as well as LIB-Static's "Collection Builder." For a more comprehensive outline of the goals of minimal computing for the GO:DH Minimal Computing Group, see [Gil 2015] and [Sayers 2016]. There is also a growing interest in the use of static collections within libraries as demonstrated by [Newson 2017], [Diaz 2018], and [Wikle, Williamson, & Becker 2020].

[11] See https://github.com/projectEndings/diagnostics and https://github.com/projectEndings/html_diagnostics.

[12] Google Web Fundamentals, https://developers.google.com/web/fundamentals/security/csp.

[13] This is not strictly true, of course — minor typographical variants are common within a single edition — but for most practical purposes we behave as though it is.

[14] While this paper does not address project management per se, it is worth noting that this model provides some significant benefits in terms of digital project management. For instance, milestones help prevent feature creep as each release requires careful and considered planning regarding project priorities; by the same token, it allows some space — when combined with Continuous Integration processing mentioned in 5.3 — for experimentation since there is no connection between the deployed site and the source data and thus no risk of breaking the existing site.

[15] *The Map of Early Modern London* retains all past static editions of the site. See "Previous Map of London Editions" for all editions.

[16] Including the version control information also makes it possible to rebuild this edition exactly as it is, if data is lost or corrupted.

[17] While redirects are often created via server configuration, static sites can implement redirects using HTTP headers and other mechanisms. See MDN, "Alternative way of specifying redirects" (https://developer.mozilla.org/en-US/docs/Web/HTTP/Redirections#alternative_way_of_specifying_redirections).

[18] Perhaps the best-known examples are FlexSearch, Lunr.js and Elasticlunr.js. None of the available alternatives provided the sophistication and scalability our sites needed.

[19] https://endings.uvic.ca/staticSearch/docs/index.html.

[20] See [Warwick 2020] for a recent historical analysis of digital humanities interface and for a discussion of the challenges presented by an incomplete or unusable archived copy of a project.

## Works Cited

**Arneil & Holmes 1998** Arneil, S., and Holmes, M. D. (1998) "Hot Potatoes: Free Tools for Creating Interactive Language Exercises for the World Wide Web", paper presented at the EuroCALL conference, Leuven, Belgium, September 1998.

**Arneil & Holmes 1999** Arneil, S., and Holmes, M. D. (1999) "Juggling hot potatoes: Decisions and compromises in creating authoring tools for the Web", *ReCALL Journal*, 11(2), pp. 12–19. Available at: https://doi.org/10.1017/S0958344000004912 and https://web.archive.org/web/20050526203034/http://www.eurocall-languages.org/recall/pdf/rvol11no2.pdf.

**Arneil & Holmes 2001** Arneil, S., and Holmes, M. D. (2001) "Hot Potatoes, History and Future", paper presented at EuroCALL Conference, Nijmegen, August 2001. Available at: https://web.uvic.ca/hrd/eurocall2001/HotPotPastFuture/PastFutureHome.htm.

**Arneil, Holmes & Newton 2019** Arneil, S., Holmes, M. D., and Newton, G. (2019) "Project Endings: Early Impressions From Our Recent Survey On Project Longevity In DH", paper presented by Greg Newton at Digital Humanities 2019 Conference, Utrecht, the Netherlands, July 2019. Available at: https://dev.clariah.nl/files/dh2019/boa/0891.html.

**Barats, Schafer & Fickers 2020** Barats, C., Schafer, V., and Fickers, A. (2020) "Fading Away... The challenge of sustainability in digital studies", *DHQ: Digital Humanities Quarterly*, 14(3). Available at: http://www.digitalhumanities.org/dhq/vol/14/3/000484/000484.html.

**Burnard 2016** Burnard, L. (2016) "How to Update Your ODD", TEI GitHub Repository. Available at: http://teic.github.io/PDF/purifyDoc.pdf (Accessed: 18 February 2019).

**Cayless 2010** Cayless, H. (2010) "Ktêma es aiei: Digital Permanence from an Ancient Perspective" in Mahony, S., and Bodard, G. (eds.) *Digital Research in the Study of Classical Antiquity*, pp. 139-150. Available at: https://www.taylorfrancis.com/chapters/edit/10.4324/9781315577210-18/kt%C3%AAma-es-aiei-digital-permanence-ancient-perspective-hugh-cayless.

**Diaz 2018** Diaz, C. (2018) "Using Static Site Generators for Scholarly Publications and Open Educational Resources", *The Code4Lib Journal*, 42. Available at: https://journal.code4lib.org/articles/13861.

**Gelaw et al. 2002** Gelaw, D., Hastings, S., and Hartman, C. (2002) "A Metadata Approach to Preservation of Digital Resources: The University of North Texas Libraries' Experience", *First Monday*, 7(8). Available at: https://doi.org/10.5210/fm.v7i8.981.

**Gil 2015** Gil, A. (2015) *The User, the Learner and the Machines We Make* [Online]. Minimal Computing: A Working Group of GO::DH. Available at: http://godh.github.io/mincomp/thoughts/2015/05/21/user-vs-learner/.

**Goddard & Seeman 2020** Goddard, L., and Seeman, D. (2020) "Negotiating Sustainability: Building Digital Humanities Projects that Last", in Crompton, C., Lane, R. J., and Siemens, S. (eds.) *Doing More Digital Humanities*. London: Routledge, pp. 38–57.

**Government of Canada 2016** Government of Canada. (2016) *Tri-Agency Statement of Principles on Digital Data Management* [Online; updated 2021]. Available at: https://science.gc.ca/eic/site/063.nsf/eng/h_83F7624E.html.

**Holmes & Takeda 2018** Holmes, M. D., and Takeda, J. (2018) "Why do I need Four Search Engines?", Proceedings of the Japanese Association of Digital Humanities Conference, Tokyo, Japan. Available at: https://conf2018.jadh.org/files/Proceedings_JADH2018.pdf#page=58.

**Holmes & Takeda 2019a** Holmes, M. D., and Takeda, J. (2019) "The Prefabricated Website: Who needs a server anyway?", paper presented by Martin Holmes at the Text Encoding Initiative Conference, Graz, Austria, September 2019. Available at: https://doi.org/10.5281/zenodo.3449196.

**Holmes & Takeda 2019b** Holmes, M. D. and Takeda, J. (2019) "Beyond Validation: Using Programmed Diagnostics to Learn About, Monitor, and Successfully Complete Your DH Project", *Digital Scholarship in the Humanities*, 34(suppl_1), pp. i100–i109. Available at: https://doi.org/10.1093/llc/fqz011.

**Holmes & Takeda 2020** Holmes, M. D. and Takeda, J. (2020) "Static Search: An Archivable and Sustainable Search Engine for the Digital Humanities", paper presented at the Digital Humanities Summer Institute (DHSI) Colloquium (#VirtualDHSI), 6 June 2020. Available at: https://doi.org/10.5281/zenodo.3883150.

**Holmes 1997** Holmes, M. D. "Web Language". Online presentation showcasing the use of JavaScript-based exercises for language teaching (1997). https://web.uvic.ca/lancenrd/martin/weblang/.

**Holmes 2017a** Holmes, M. D. (2017) "Whatever happened to interchange?", *Digital Scholarship in the Humanities*, 32(suppl_1), pp. i63–i68. Available at: http://dx.doi.org/10.1093/llc/fqw048.

**Holmes 2017b** Holmes, M. D. (2017) "Selecting Technologies for Long-Term Survival", paper presented at the SHARP Conference 2017: Technologies of the Book, Victoria, BC, Canada, June 2017. Available at: https://github.com/projectEndings/Endings/raw/master/presentations/SHARP_2017/mdh_sharp_2017.pdf.

**Holmes et al 2019** Holmes, M. D., Fralick, K., Fukushima, K. and Karlson, S. (2019) "How we tripled our encoding speed in the Digital Victorian Periodical Poetry project", paper presented at the Text Encoding Initiative Conference, Graz, Austria, September 2019. Available at: https://zenodo.org/record/3449241.

**InterPARES 2013** InterPARES 3 Team Canada (International Research on Permanent Authentic Records in Electronic Systems). (2013) *Case Study 09: Alma Mater Society of The University of British Columbia: Policies and Procedures for Web Site Preservation*. Available at: http://www.interpares.org/ip3/display_file.cfm?doc=ip3_canada_cs09_wks02_action_25_v1-3.pdf.

**Jakob Nielsen 1999** Nielsen, Jakob (1999) *Graceful Degradation of Scalable Internet Services*. Nielsen Norman Group Website. Available at: https://www.nngroup.com/articles/graceful-degradation-of-scalable-internet-services/.

**Jakobsen & Orlandi 2016** Jakobsen, J., and Orlandi. C. (2016) "On the CCA (in)security of MTProto", *SPSM '16: Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices, October 2016*, pp. 113–116. Available at: https://doi.org/10.1145/2994459.2994468.

**Jim Nielsen 2021** Nielsen, Jim. (2021) *Web Languages as Compile Targets*. Jim Nielsen's Blog. Available at: https://blog.jim-nielsen.com/2021/web-languages-as-compile-targets/.

**Molloy 2011** Molloy, L. (2011) *Oh, the Humanities! A Discussion about Research Data Management for the Arts and Humanities Disciplines*. JISC MRD - Evidence Gathering. Available at: https://web.archive.org/web/20141218214525/http://mrdevidence.jiscinvolve.org/wp/2011/12/16/oh-the-humanities-a-discussion-about-research-data-management-for-the-arts-and-humanities-disciplines/.

**Morgan 2021** Morgan, P. (2021) *Further thoughts on collectivity*. Paige Morgan. Available at: https://web.archive.org/web/20221218222956/http://blog.paigemorgan.net/articles/21/further-thoughts.html.

**Morreale 2019** Morreale, L. (2019) "Medieval Digital Humanities and the Rite of Spring: Thoughts on Performance and Preservation". Available at: https://doi.org/10.34055/osf.io/7p2t6.

**Newson 2017** Newson, K. (2017) "Tools and Workflows for Collaborating on Static Website Projects", *The Code4Lib Journal*, 38. Available at: https://journal.code4lib.org/articles/12779.

**Rinaldi 2015** Rinaldi, B. (2015) *Static Site Generators*. O'Reilly Media, Inc.

**Sacchi 2015** Sacchi, S. (2015) "What Do We Mean by 'Preserving Digital Information'? Towards Sound Conceptual Foundations for Digital Stewardship", PhD thesis, University of Illinois at Urbana-Champaign. Available at: https://doi.org/10.7916/D8WW7GMK.

**Sayers 2016** Sayers, J. (2016) *Minimal Definitions*. Minimal Computing: A Working Group of GO::DH. Available at: http://go-dh.github.io/mincomp/thoughts/2016/10/02/minimaldefinitions/.

**Takeda & Lines 2019** Takeda, J. and Lines, S. (2019) "Using Github and its Integrations to Create, Test, and Deploy a Digital Edition", paper presented at the Text Encoding Initiative Conference, Graz, Austria, September 2019. Available at: https://gams.uni-graz.at/o:tei2019.174.

**Thieberger 2016** Thieberger, N. (2016) "What remains to be done – Exposing invisible collections in the other 7000 languages and why it is a DH enterprise", *Digital Scholarship in the Humanities*, 32(2), pp. 423–434. Available at: http://dx.doi.org/10.1093/llc/fqw006.

**Unsworth 1997** Unsworth, J. (1997) "Documenting the Reinvention of Text: The Importance of Failure", *The Journal of Electronic Publishing*, 3(2). Available at: https://doi.org/10.3998/3336451.0003.201.

**Viglianti 2017** Viglianti, R. (2017) "Your Own Shelley-Godwin Archive: An off-line strategy for on-line publication", paper presented at the Text Encoding Initiative Conference, Victoria, BC, Canada, November 2017. Available at: https://hcmc.uvic.ca/tei2017/abstracts/t_126_viglianti_shelleygodwin.html.

**Warwick 2020** Warwick, C. (2020) "Interfaces, Ephemera, and Identity: A Study of the Historical Presentation of Digital Humanities Resources", *Digital Scholarship in the Humanities*, 35(4), pp. 944–971. Available at: https://doi.org/10.1093/llc/fqz081.

**Wikle, Williamson, & Becker 2020** Wikle, O., Williamson, E., and Becker, D. (2020) "What is Static Web and What's it Doing in the Digital Humanities Classroom?", *dh+lib*. Available at: https://dhandlib.org/2020/06/22/what-is-static-web-and-whats-it-doing-in-the-digital-humanities-classroom/.