A Renewed Look at Greenstone: Entering the Third Decade

Bainbridge, David; and Witten, Ian H.

Abstract: The Greenstone Digital Library Software has helped spread the practical impact of digital library technology around the globe. As Greenstone enters its third decade, this article takes a renewed look at its development, the challenges that have been faced, and the lessons that have been learned in deploying a comprehensive open-source system for the construction of digital libraries internationally. In particular, we outline architectural changes in the software that have occurred over time, and highlight how the user interfaces have evolved to provide a more immersive, interactive user experience, made possible through advancements in the underlying web technologies.

Index Terms: Digital Libraries, Interface Design, Software Architecture

1. *Introduction*

It has been two decades since the name Greenstone was adopted for the emerging code-base that was, at the time, being developed under the auspices of the New Zealand Digital Library Project, and a decision made to distribute it under the GNU General Public License. Over this time the project's overarching aim has remained unchanged: helping spread the practical impact of Digital Library technology through the development and deployment of a comprehensive open-source system for the construction of digital libraries internationally.

Manifest through versioned releases of Greenstone, the software is not a digital library *per se*, but a tool that empowers others to build and distribute fully-searchable, metadata-driven digital collections of their own content, accessed from the web or removable media such as a USB thumbdrive or DVD. Examples range from OCR'd document collections focused on humanitarian development to audio- and video-based oral histories by indigenous peoples; from digitized Armenian rare books to born-digital archaeological reports from site digs; from historic early editions of musical compositions to institutional repositories that capture and distribute academic outputs. For further examples see:

www.greenstone.org/examples.

Although the NZDL Project, with associated software, stretches back to the mid-1990s, Greenstone itself was registered on SourceForge in September 2000. Since then, the software has been downloaded from practically every country in the world. SourceForge statistics record 220 countries, although this is an accumulated figure—some no longer exist, and others have emerged. The top five by total download count are (in order) India, USA, Vietnam, Ethiopia, and Argentina. Greenstone recently topped the 1 million download mark, including binary distributions, source code and documentation.

The drive for internationalisation was stimulated by partnerships with UN agencies—in particular UNESCO—and the Belgium-based Human Info NGO. Thanks to an online community of volunteer translators, Greenstone's web interface (which we refer to as the *Reader's Interface*) has been translated into 60 languages. The graphical interface for collection building and maintenance used by digital librarians, curators, and other content managers (the *Librarian Interface*) is available in 23 languages. At the time of writing the mailing list contains over 800 users, three-quarters of whom are on the "users" list and the remainder on the "developers" list.

On Greenstone's tenth birthday we reflected on what had been achieved in *A Retrospective Look at Greenstone: Lessons from the First Decade* [8]. The present article provides an update by describing the second decade of development and summarising key accomplishments. This is largely reflected in the move to Greenstone 3, which utilises the same collection-building code base as the earlier Greenstone 2 but includes a complete rewrite of the runtime system to take advantage of newer web standards.

We begin by detailing how the software has evolved, noting in particular pertinent architectural changes. Next we compare and contrast the end-user's experience of Greenstone 3's Reader's Interface with its forerunner. We then highlight two new features—the Document Editor and Greenbug—designed specifically to support the digital librarian when undertaking maintenance and management work *in situ* in the digital library. Finally we summarize key lessons learned.

Manuscript received on October 28th, 2019.

David Bainbridge and Ian Witten are with the New Zealand Digital Library Project, Department of Computer Science, University of Waikato (e-mail: {davidb,ihw}@waikato.ac.nz).

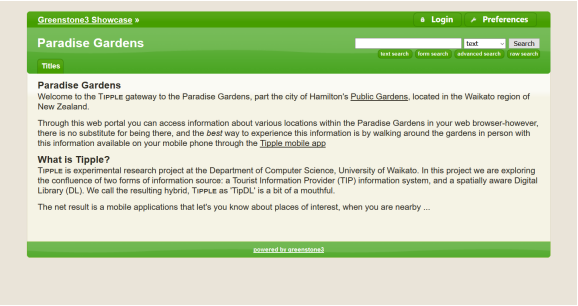

Figure 1: The *About this Collection* page for the Paradise Gardens collection using the default Greenstone 3 interface.

Figure 2: The same *About this Collection* page using Greenstone 3's halftone interface.

2. *From Greenstone 2 to Greenstone 3*

The core architectural decisions for the Greenstone 2 software were made in the late 1990s, which was very much a formative period of the web. Many web technologies that are commonplace today were in their infancy—or non-existent. We frequently ended up implementing something from scratch that got the job done.

A case in point is the configuration file for a collection. Back then, the XML format had yet to be invented! We created a bespoke textual file format in which each line began with one of a list of keywords that dictated how remaining text on the line was to be interpreted. Although we wrote detailed documentation describing what was needed to create collection configuration files and how to specify things in this admittedly arcane format, users were left to their own devices in entering the information in a text editor. The consequences of error were severe and, often, incomprehensible.

Today, an XML format accompanied by an appropriate XML Schema allows a range of text editors to support the creation of valid configuration files, including highlighting syntax errors. We return to this point in Section 2.5.

Having seen how web technologies have developed, our "roll-you-own" approach to the core components of the Greenstone 2 architecture seems appropriate for the time. Summing up this period of the project, we developed a suite of components and combined them into web-based workflows that are recognisable in today's web technologies—with the caveat that our inventions were form-fitted to the specific functionality needed in a digital library.

As the project moved forward, we reported on the burgeoning development of Greenstone 3 in [8]. Introducing research-led features into Greenstone prompted user reports of unforeseen problems on the discussion list, so Greenstone 3 was conceived as a purely research-based project in which to pursue new avenues. Built using the more established web technologies, backwards-compatibility was possible, but was not provided by a consumer-focused point-and-click graphical installer: a non-trivial level of IT skills was required.

Over time, however, the demands of maintaining two versions of the software outstripped our resources, and in *A Retrospective Look at Greenstone: Lessons from the First Decade* we noted that a decision to make Greenstone 3 the flagship version of the software had just been taken. Before that point, Greenstone 2 was promoted as the production version—with a graphical installer and comprehensive tutorials—and Greenstone 3 was the "bleeding edge" research-led version. Over a gradual transition process, Greenstone 2 was marked as deprecated, with releases focused solely on bug fixes, and Greenstone 3 built up to be the production version. The transition is now complete.

2.1 *Greenstone 3 Software Overview*

A graphical installer for Greenstone 3 was developed that set up a Tomcat web server with a servlet-based runtime system, in contrast to the Apache *httpd* web server with CGI scripts used by Greenstone 2. This required rewriting all 30 existing tutorial exercises for Greenstone 3. Developing the new installer helped us consolidate and generalise the in-house "release-kit" code that had been written to automate the generation of a software release, and Greenstone 3's backwards compatibility guaranteed that all the tutorial exercises could be expressed within the new architecture.

Strong adoption of mainstream web technologies has the enormous side benefit of being able, in our user instructions, to focus on the requisite steps and the reasons for them, rather than getting caught up in explaining the idiosyncrasies of in-house syntax. The number of tutorials has been increased to around 40 to reflect new features—such as changing the look and feel of the Reader's Interface by creating new graphical styles using the interactive (point-and-click) JQuery-UI Theme Roller.[1]

Figures 1 and 2 illustrate how different styles can be applied to the same digital library content. Figure 1 shows the *About this collection* page for the Paradise Gardens collection using the default Green-

[1] https://jqueryui.com/themeroller/

stone 3 Reader's Interface. It adopts the default sans-serif font of the user's web browser and, in keeping with the software's name, uses a mid-to-dark green background colour for the header and footer of each page served up by the system. Figure 2 shows a different look for the same page, with a bespoke colour palette and text display controlled by web fonts. Moreover, it includes Twitter and Facebook links, encouraging visitors to share what they are doing with others through social media.

To reconcile Greenstone 3's original aim of supporting research-led development with its role as the production environment, the concept of "extensions" was introduced. Extensions operate at the file-system level: the digital library set-up procedure looks for sub-directories of an *ext* directory and accesses their content as appropriate. Extensions can manipulate environment variables so that their functionality remains *in situ* on the file system (common in the case of build-time extensions), or else provide 'install' and 'uninstall' directives that modify the core code base (common in runtime extensions). In the latter case, recompilation of Greenstone is typically required.

The best way to see the differences between Greenstone 3 and 2 is through examples of the interface in use, which we provide in Sections 2.2–2.4. To appreciate these, it is useful to know something about the underlying software architecture, in particular the role of XSL Transforms (XSLT) [5].

A Greenstone 3 digital library is conceived as a modular network of services, where XML messages are passed around as a result of user-initiated tasks or interactions—typically performed in a web browser—or system-initiated tasks—such as automated testing. This gives the architecture sufficient versatility to meet research needs. The production version of the software is delivered by priming the default installation with services that implement the basic functionality of search, browsing, document view and user preferences, as in Greenstone 2.

To respond to web-browser user interactions, a "Receptionist" module applies XSLT transforms to XML generated messages returned to the browser. This transforms the message into XHTML, which the browser then displays. The transforms are determined by the digital librarian, which is how presentation and functional changes are managed. It is also possible to stipulate that the final XML message is expressed in JSON format [7], which is useful when delivering Web 2.0-configured interfaces. Further details of Greenstone 3's software architecture are described in [3].

2.2 *Reader's Interface*

The Reader's Interface is the web browser interface for end users of Greenstone digital library collections, and is quite distinct from the Librarian Interface, used by librarians, archivists and other managers of digital content to develop and maintain Greenstone collections. Greenstone 3's Reader's Interface is a complete reimplementation of the earlier version that maintains the core features of search, browse, document view, and user preferences, but places greater emphasis on presenting users with an immersive, interactive experience.

To convey a sense of the changes, we compare the Greenstone 2 and 3 versions of the Reader's Interface to the *Niupepa: Newspapers in Māori* collection. This is a collection of historic newspapers published between 1842 and 1932, primarily for a Māori audience. Based on a microfiche collection created by the Alexander Turnbull Library (part of New Zealand's National Library), it contains 17,000 digitized pages that have been OCR'd—mostly manually corrected by hand to fix transcription errors—to support full-text searching. 70% of the collection is written solely in Māori, 3% in English, and the remainder is bilingual. There are three main types of niupepa: government sponsored, Māori initiated, and religious.

Figure 3 shows Greenstone 2's *About this collection* page. It conveys in general terms what users can expect to find in the digital resource. In the case of *Niupepa* it provides the sort of details we have just given, and more.

Figure 4 shows the Greenstone 3 version. It contains the same information as its predecessor, but there are clearly visible differences—principally the page width and the elements displayed in the page header. It used to be common practice to expand web pages to the full width of the browser window—typically the width of the user's screen. But desktop screens have become wider, leading to very long lines of text, which is known to impact readability. Today's websites usually limit the page width, which is easy to control using Cascading Style Sheets—a web standard that gained traction starting around 2001 [4] and is now used throughout Greenstone 3.

The headers of Figures 3 and 4 both give access to the core features of search, browsing, search, and user preferences. However, with advancements in cross-browser support for visual and typographical effects, it is no longer necessary to use custom-generated images to achieve the desired visual effect as Greenstone 2 did—images that had to be generated for each language supported by the interface.

In Greenstone 3 the different ways to access a collection appear as a row of tabs, and the header includes a quick-search box, a common trend in contemporary website design. A button below the quick search box takes the user to an advanced search page.

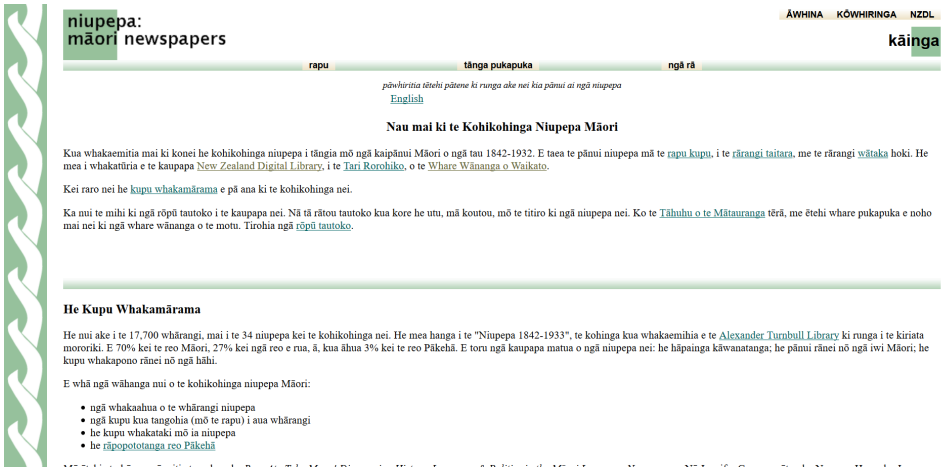

Figure 3: The *About this collection* page in Greenstone 2 for the *Niupepa: Newspapers in Māori* collection.

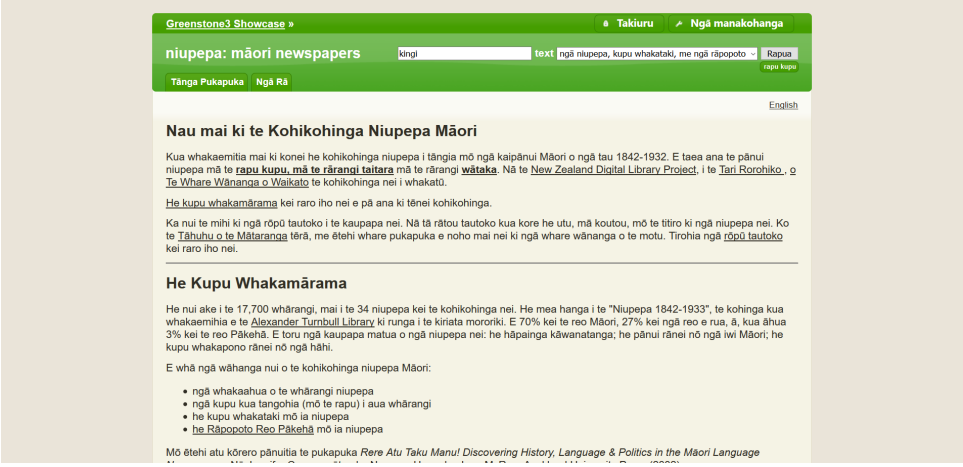

Figure 4: The *About this collection* page in Greenstone 3 for *Niupepa: Newspapers in Māori*.

Figure 5: Top-level browsing by series in Greenstone 2 for *Niupepa: Newspapers in Māori*.

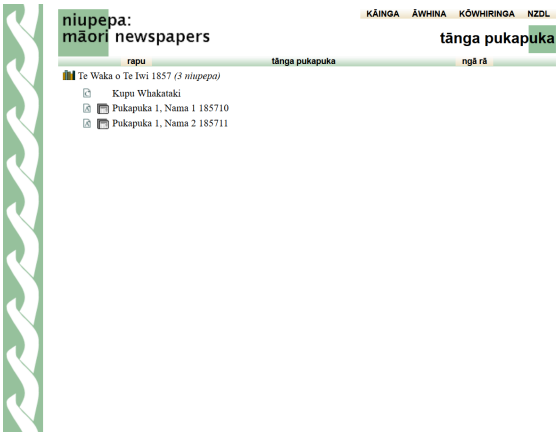

Figure 6: Drilling in to a particular newspaper title when browsing by series in Greenstone 2 for *Niupepa: Newspapers in Māori*.

2.3 *Browsing*

To illustrate the more immersive user experience that Greenstone 3 provides, consider the *browse by serial* feature of *Niupepa*. This shows all newspaper titles—such as *The Maori Messenger – Ko te Karere Maori* and *Te Waka Maori o Ahuriri*—sorted alphabetically, from which one can explore particular titles to determine what editions of that paper the collection contains, listed by volume and number. Figures 5 and 6 show how the mechanism used to work, and Figure 7 shows the new approach.

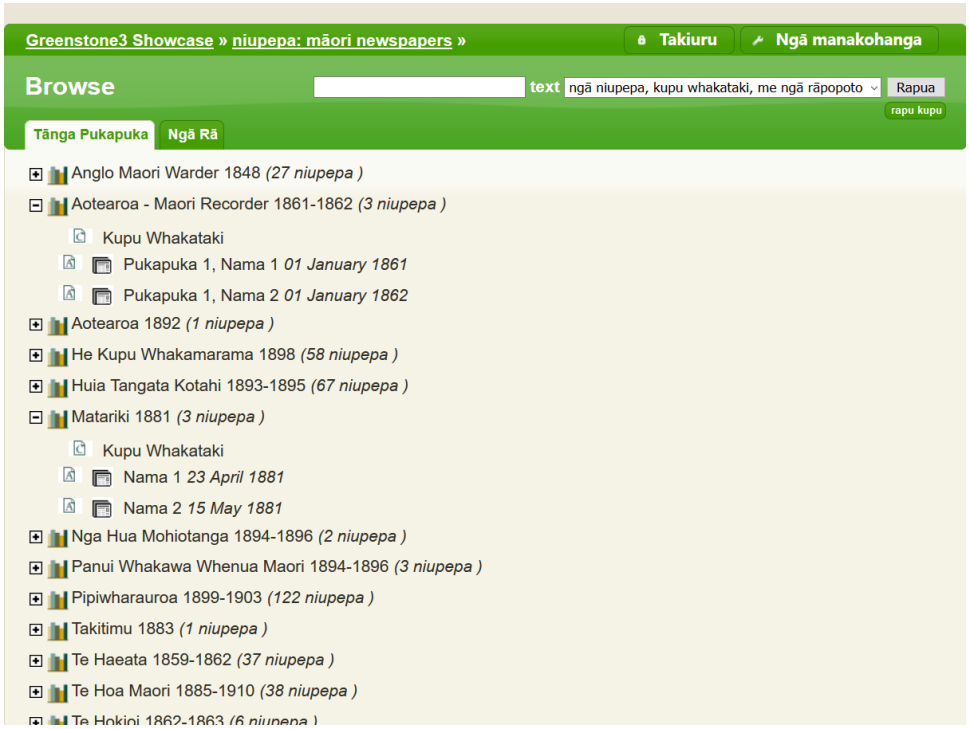

Figure 7: AJAX-based Interactive browsing by series in Greenstone 3 for *Niupepa: Newspapers in Māori*.

In Greenstone 2, a new page is loaded every time a user interacts with the browsing hierarchy. Figure 5 shows the list of titles, and clicking one—*Te Waki o Iwi*, for example—to view its editions causes the browser to load the page in Figure 6, which displays the newspaper's title, and, beneath and slightly indented, a commentary on that paper (if available) and the editions published (two, in this case).

With the emergence of AJAX interaction with the Document Object Model (DOM) in Web 2.0 [2], the browsing feature has been redesigned to provide a better overview of the information and reduce the volume of data transmitted over the network.

In Figure 7 we join the user partway through their interaction. Initially they see the same top-level "bookshelf" icons for newspaper titles as before. But when they click on one, instead of reloading the entire page as Greenstone 2 does (retransmitting all the header, footer and page formatting information), Greenstone 3 makes an AJAX call for just the items under that bookshelf, which it adds to the page using DOM manipulation. Users can still see the full range of titles, and—as has occurred in the Figure—continue their exploration of other titles, opening up further bookshelves, while retaining an overview of all titles.

Clicking an opened bookshelf returns it to the closed state. The browser remembers which bookshelves have been opened, so that if a user further explores another newspaper title—which causes a full page load—and then navigates back, any previously opened bookshelves are restored. Since the AJAX calls need less network data to operate than in Greenstone 2, the new interface feels more responsive. The results of these AJAX calls are cached, so that if the user, browsing around, opening and closing folders, ever returns to a previously-opened bookshelf, the interface skips the network call and shows what it has already stored for that node.

Another (unconnected) aspect of the new interface is the use of "breadcrumbs" [6], a feature designed to help users keep track of where they are in a website—in this case a Greenstone library. The breadcrumb is located in the page header, shown at the left of the very first line of text. In Figure 7 it is *Greenstone3 Showcase » niupepa: māori newspapers*, the name of the library being visited (*Showcase*) followed by the collection being accessed. If a user were to view one of the newspapers the breadcrumb would change to *Greenstone3 Showcase » niupepa: māori newspapers » Tuhinga* (or, in the English interface, *Greenstone3 Showcase » niupepa: māori newspapers » Document*). This trail of breadcrumbs occurs throughout the digital library.

2.4 *Document View*

Figures 8–11 illustrate how the document view has changed between the two versions of Greenstone, again using the *Niupepa* collection as the example. In Greenstone 2, the "document view" for an OCR'd scanned image provides buttons that display the document's text (Figure 8), a preview image (Figure 9), and the full-size version (Figure 10). Users can click forward and backwards to access adjacent pages, or enter a page number into an input box. As when browsing, each of these interac-

Figure 8: The Greenstone 2 text view for a page of *Niupepa: Newspapers in Māori*.

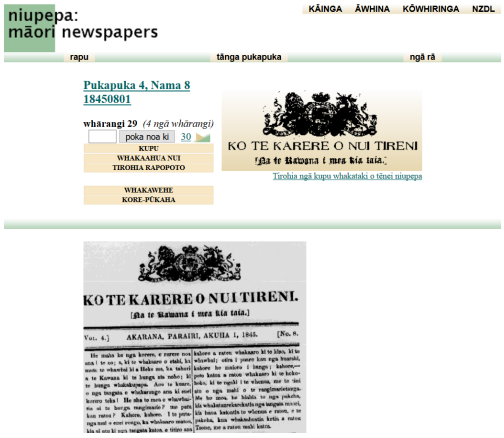

Figure 9: The preview image view of the page in Figure 8.

tions invokes a page reload.

Greenstone 3 brings all this—and more—together in a single integrated view. Figure 11 shows the main scanned-image document view. On the right is a page-preview table-of-contents area, enlarged in Figure 12, which is styled in the fashion of popup preview windows often shown when printing a document. When the user accesses a newspaper in the collection, thumbnail images of all its pages are (asynchronously) loaded. Clicking on one displays the page at screen resolution in the main interface area; alternatively a sequence of contiguous or non-contiguous page numbers (e.g., 1, 6-9) can be entered into the text box underneath the thumbnails and the main page dynamically updates appropriately.

The controls above the preview area in Figures 11 and 12 determine whether the page text, the screen-view image, or both, are shown. A magnifier is included that operates on the screen-view image as the cursor moves over it; alternatively, a full-screen view can be obtained (using the canvas-on-easel icon) that occupies the full browser window rather than the limited width used in the other displays, so

Figure 10: The full image view of the page in Figure 8.

that users can see as much of the high-resolution image as possible.

The thumbnail images are displayed horizontally but the screen-resolution/text-displayed pages they relate to are organized vertically. This part of the interface operates in the same way as the hierarchical browsing view described above. AJAX calls are made for just those parts of the document required for display, again increasing the interface's responsiveness. As before, caching is used to avoid making network calls for repeat visits to parts of the document.

As users continue to explore pages of a newspaper the length of the vertical display grows. To avoid having to repeatedly scroll up in order to interact with the page-preview control area, a "tear-away" feature is provided that removes it from the vertical page display and attaches it to the upper right-hand side of the page. If users find that the page-preview control area encroaches on their screen space they can minimise it by clicking the close icon in the top right-hand corner of the control panel.

2.5 *Librarian Interface*

Greenstone 3 retains Greenstone 2's "Librarian interface" (GLI), a Java application that creates, develops, and helps maintain a digital library. GLI has been re-coded to utilize the new XML collection configuration file format. Also, the *Format* tab, through which presentation details for the collection can be specified, has been updated. In Greenstone 3 format statements are expressed using XML syntax, and the graphical user interface has been changed to use the *RSyntaxTextArea*[2] component rather than Greenstone 2's generic freeform *TextArea*, bestowing the syntax error highlighting feature that we mentioned earlier as a key benefit of using new standards.

GLI's purpose is to provide a graphical user in-

[2]https://bobbylight.github.io/RSyntaxTextArea/

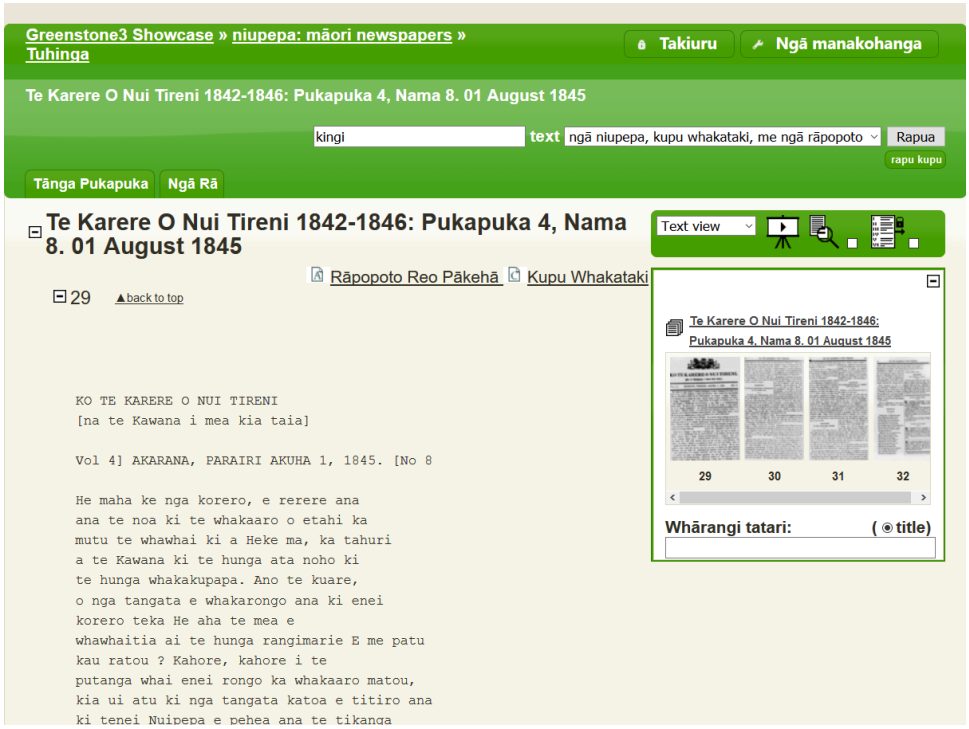

Figure 11: Greenstone 3's main scanned-image document view.

Figure 12: Enlarged excerpt of Figure 11 showing the interactive preview area.

terface for controlling all aspects of the design of a digital library collection. This is particularly useful in the early stages of collection development, but becomes cumbersome when performing on-going maintenance. For example, if a librarian notices an error in a document's metadata when browsing on the web, to correct it they must (i) start up GLI and (ii) relocate the offending document through that in-

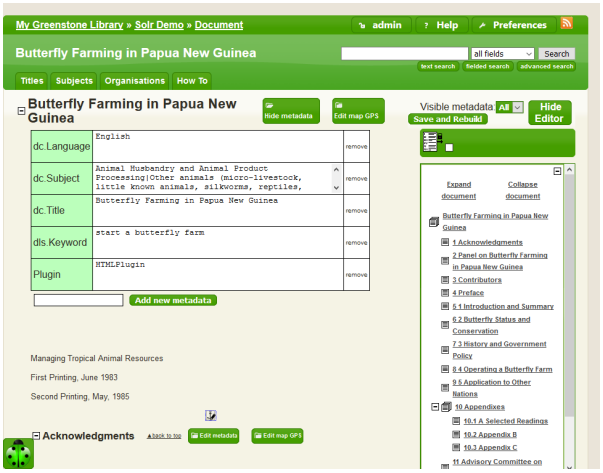

Figure 13: The in situ document text and metadata editor in action.

terface, which only shows a file-system view, before (iii) accessing GLI's *Enrich* tab to correct the error and update the digital library.

To rectify this, Greenstone 3 contains editing capabilities that operate within the browser, giving direct ways to manage and maintain collections. The librarian mentioned above clicks on a *Login* button (*Takiuru* in the Māori interface shown in Figure 11) and accesses the "Document Editor" shown in Figure 13. This wiki-like editor gives authenticated users access to sections of the document that they can edit and save. In addition, for each section they can click an *Edit Metadata* button to reveal a table of the metadata assigned to that part of the document.

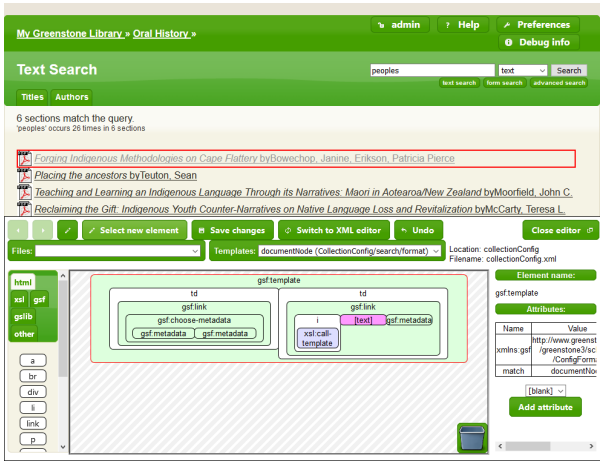

Figure 14: Greenbug in action.

They can correct values, remove them, or add new ones, and save all this to the digital library.

The Solr[3] indexing tool has been included in Greenstone 3 as an option for document ingest, and this supports spatial search. GPS metadata embedded in documents are extracted and used to provide a proximity-based search service. In the Document Editor, an *Edit Map GPS* button appears beside the edit metadata button, which invokes a Google Maps component through which regions and GPS markers can be entered.

Figure 14 shows another type of *in situ* editing that Greenstone 3 supports, in this case to control the overall structure and functional features the library provides. It was inspired by Firebug, a Firefox extension for web developers who wished to inspect the underlying structure and state of web pages—a capability that is now standard in all major web browsers. Web inspectors help developers trace the reason why something is not displaying the way it should, and allow them make changes to the underlying syntax and immediately see their effect.

The "Greenbug" editor [1] allows one to inspect the state of pages served up by the digital library software. As in Firebug, the user starts by selecting an element. As they move around a library web page, a highlighter box follows the cursor, snapping to elements that they hover over. For example, the upper portion of Figure 14 shows a Greenstone search page with the highlighter box snapped over the top item in the list of search results. The lower portion gives an interactive, editable representation of the structure that has produced that part of the page. Whereas Firebug shows the HTML elements that are responsible for the selected part of the page, Greenbug shows the underlying XSLT elements. These elements can be edited, and the edits take effect immediately in the page displayed above. Moreover, Greenbug goes further than Firebug (and other web inspectors) by allowing users to save their changes. This is possible because it is closely integrated with the underlying Greenstone 3 system. By first ensuring that users are authenticated, the Greenbug code is granted access to more trusted parts of the service-based API—such as the ability to update the XSLT files used by the digital library.

[3]https://lucene.apache.org/solr/

3. *Insights and Lessons Learned*

What has been learned in Greenstone's second decade?

Hits and Misses. A recurring criticism was that all Greenstone 2 collections are displayed on the same top-level page. Unrelated sets of collections can only be kept separate by installing a new instance of Greenstone on the same server. Greenstone 3 addresses this issue by introducing the notion of *sites*, which gives precisely this ability within a single Greenstone installation. A further layer of abstraction, *interfaces*, allows the curator of the digital library installation to select or develop different look-and-feels, which can then applied to individual sites. These new capability have proved to be extremely popular with librarian users—a hit!

Another criticism was the cognitive effort required by developers to learn how to control the presentation and structure of their digital library installation. Format statements and macro files in Greenstone 2 were the bane of their lives! We developed both of these from scratch, and had to provide a great deal of documentation and tutorial support, in the form of exercises, to educate users. Greenstone 3 unifies these two aspects through the use of the XSLT web standard. However, we now get complaints that this is more verbose than the original format statements and macro files, and the inheritance mechanism that operates across collection, site and interface makes it hard to find the appropriate XSLT rule to edit. The introduction of Greenbug should alleviate this concern. Hit or miss?—time will tell.

Learning Greenstone. We previously developed an extensive set of tutorial exercises, which are the principal means by which people learn how to use the software. However, during live tutorials it is apparent that participants often struggle to extrapolate from the exercises to meet their own needs. We now believe the cause to be the "point-and-pick" nature of the tutorials, which was originally adopted to enable people to work independently and at their own pace. To rectify this we plan to append open-ended sections to each tutorial that supply a fresh set of files and metadata, just like what the user has been working with, accompanied by more general instructional steps. For example: *Build the HTML files into a collection with a Browse by Title feature. For HTML pages where the automatically extracted title does not appropriately reflect the document, manually assign a better one.*

Reluctance to change from Greenstone 2. Organisations and users who have established Greenstone 2 installations and operate them with minimal (or no) funding tend to stick with what they have working. While we understand the reason—if it ain't broke, don't fix it—we regret that these users cannot take advantage of the significant new features that Greenstone 3 provides. Luckily, an unanticipated consequence of the Greenstone 3 decision to continue to use and extend the Perl-based Greenstone 2 code for ingesting documents and metadata (such as enhanced processing of PDF files) is that improvements here also benefit those who have stuck with Greenstone 2.

Documentation bottleneck. It is always difficult to provide an appropriate level of documentation for users with a wide range of skill sets, and this is something that our small developer team struggles with. To help offset this, from the very beginning of Greenstone 2 and throughout the development of Greenstone 3 we have consciously designed and used data structures that are self-documenting.

This works well at the micro level. For example, when a GLI user is deciding how PDF documents being ingested into a collection should be processed —should keywords, for example, be automatically assigned based on the full-text of the document?— all available options available are displayed, along with tooltips translated into the language GLI has been set to use. All this is programmatically generated by Greenstone.

Concerning configuration control at the macro-level—such as whether or not a map view is shown when browsing by title, for example—we have begun to consciously include what we refer to as "configuration setting breadcrumbs." This extends the breadcrumbs notion introduced earlier to deliberately incorporate settings in a configuration file that are redundant but are useful for others to find—for instance, including an XML element that explicitly assigns a default value to a variable even though it is not necessary for this value to take effect in the code; or including a commented-out block of configuration file syntax that achieves an effect different to the default behaviour, to demonstrate possible alternatives. This approach complements the idea of self-documenting code in enabling self-discovery.

4. *Conclusion*

We have reviewed developments and experiences that have occurred in the Greenstone digital library software's second decade, complementing our previous article on the first decade [8]. New statistics on the user base highlights Greenstone's increased international reach, notably the number of countries where the software has been downloaded (220) and the number of languages the interface has been translated into (60). We have described some of the technical accomplishments and interface advances in Greenstone 3, which has become both the flag-ship production version of the software and the framework in which we conduct our research. Finally, we discussed some of the lessons learned.

In terms of the wider picture, we still struggle with the tension between research-led tools and production delivery. We have had to curtail the routine inclusion of novel tools developed in our research programme into the Greenstone code base, because of the burden it places on the development team in supporting multiple operating systems. Instead, we now favour the development of web-based demonstrations embedded within Greenstone 3 that show off its versatile software architecture, postponing any decision about bringing them into the distributed version pending an assessment of the level of interest in the result.

References

[1] D. Bainbridge, S. J. McIntosh, and D. M. Nichols. Greenbug: a hybrid web-inspector, debugger and design editor for greenstone. In *Proceedings of the 13th ACM/IEEE-CS joint conference on Digital libraries*, pages 449–450. ACM, 2013.

[2] O. Campesato and K. Nilson. *Web 2.0 Fundamentals: With AJAX, Development Tools, and Mobile Platforms*. Jones & Bartlett Learning, 2010.

[3] K. Don. Greenstone3: A modular digital library. *Online: http://www.greenstone.org/docs/greenstone3/manual.pdf*, 2006.

[4] J. Hoffman. A look back at the history of CSS. *CSS Tricks*, 2017.

[5] Z. L. Rendon and J. R. Gardner. *XSLT and XPATH: a Guide to XML Transformations*. Prentice Hall, 2001.

[6] B. L. Rogers and B. Chaparro. Breadcrumb navigation: Further investigation of usage. *Usability News*, 5(2):1–7, 2003.

[7] C. Severance. Discovering Javascript Object Notation. *Computer*, 45(4):6–8, 2012.

[8] I. H. Witten and D. Bainbridge. A retrospective look at Greenstone: Lessons from the first decade. In *Proceedings of the 7th ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '07, pages 147–156, New York, NY, USA, 2007. ACM.