

On the need for open teaching on the JamStack

William Billingsley

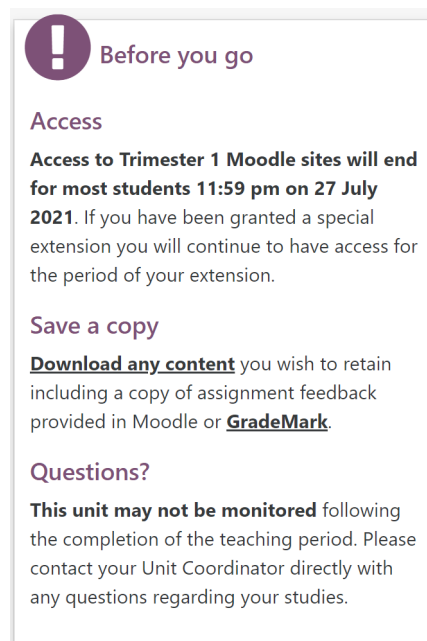
University of New England

Typical university Learning Management Systems (LMSs) place an enrolment paywall between students and the content within a unit. This has the effect not only of preventing access from potential students, but also of locking past students out from accessing updated materials as the subject develops over subsequent years to their enrolment. In this and many regards, the mechanisms by which academics can produce and publish content face limitations that open source software documentation sites do not. This provocation paper describes some of these limitations and gives an overview of the *JamStack* – common techniques that have developed within the software development community that allow convenient self-publishing of sites and materials. The paper then gives a brief introduction to *Doctacular*: a course-oriented static site generator that is under development (but already used for two live sites) to bring JamStack-style publishing to academic course materials

Keywords: Jamstack, static site generator, open educational resources.

Introduction

As I write this paper, any student opening the Learning Management System (LMS) that my university uses for my teaching units would be greeted with the notice in Figure 1 shown in the right sidebar of the unit site. Across many universities, similar notices are displayed on unit sites as the cohorts roll over to new trimesters and this is perhaps a common complaint. From the perspective of a software developer, however, each statement in the figure could be also seen as a specific criticism of how an aspect of university education currently works.



! Before you go

Access

Access to Trimester 1 Moodle sites will end for most students 11:59 pm on 27 July 2021. If you have been granted a special extension you will continue to have access for the period of your extension.

Save a copy

Download any content you wish to retain including a copy of assignment feedback provided in Moodle or **GradeMark**.

Questions?

This unit may not be monitored following the completion of the teaching period. Please contact your Unit Coordinator directly with any questions regarding your studies.

Figure 1. A standard notice displayed at the end of the teaching period, which can also be viewed as identifying three user experience issues with common university practices

“Access to Moodle sites will end for most students”

Issues around Open Education Resources (OERs) are often discussed in terms of the cost of access to textbooks before students enroll (Hilton et al., 2014; McGowan 2020; Pitt et al., 2020). Opportunities for sharing knowledge and the remixing of content between educators through open education resources, open educational practices, and open scholarship are also often discussed (Coughlan *et al.*, 2013; Mishra 2017; Luo 2020). However, there is also a practical user experience issue with closed teaching sites that is less often discussed. For those students who did enroll (and either paid or were funded publicly to study a subject), access is removed when their studies in the subject are complete. This prevents those students from updating their skills as a subject develops over subsequent teaching periods using the educational materials of the institution.

For example, consider one of my own units which teaches the Scala programming language. In 2020, this taught the then-current version of the language (2.13). In 2021, a major new version of the language was released, Scala 3, with significant new features and changes to its syntax. If I were to follow university guidelines and build materials that were primarily hosted within my institutional LMS, no 2020 student who studied Scala 2 would have access to my 2021 materials teaching Scala 3, as they would all sit once again behind the university’s enrolment paywall. Even if a hypothetical student were to be willing to pay to do so, they would not be able to: if the university student management system says you have already taken this unit and it is not a repeatable unit, you cannot enroll again.

This stands in contrast to the educational practices of open source software. For a Scala developer reading the programming language’s documentation and materials, not only do they have constant access to the most recent version, the language maintainers also provide materials describing the updates: for instance, migration guides, version highlights, and change logs. The same is true for many other popular open source projects.

“Download any content you wish to retain”

The advice that students should download content as a retention strategy is often posted under the assumption that materials can be isolated, downloaded, and work in a disconnected environment (e.g., PDF files and videos). This is not necessarily true of interactive learning environments and materials, which might require software or an ongoing connection to live services to operate.

The concept of *bit rot* is a well-known phenomenon in the computing industry (Cerf 2011). Without access to the hardware, software, or services that are needed to interpret and use it, retaining the raw content data might be of little use. For example, consider a drawerful of antique games console cartridges with no console to play them on: the content is retained, but unusable. With cloud computing, the loss of access to a necessary service can likewise render data effectively unreadable and unusable. The more digitally interactive education becomes, the more it encounters the problems of digital preservation – an area in which researchers have long grappled with how to preserve digital objects (e.g., Galloway 2004), the software that interprets them (e.g., Matthews et al., 2010), and the environments software executes within (e.g., van der Hoeven et al., 2005).

The use case of a student wanting to retain their teaching materials in a working state after their university account closes may, of course, be more limited in scope. If we assume course materials will be updated from year to year, then we may be able to assume that students would wish to retain the original version of their materials in a working state for a time but not indefinitely and that they may tolerate some loss of functionality. Nonetheless, retaining content and retaining the software the content runs within are clearly linked issues.

“This unit may not be monitored”

When a teaching period is at an end, as well as the students moving on from the LMS site, so too do the teaching staff. Moreover, within many current LMSs, so do the materials. Unit sites for the next offering are often created by cloning the previous offering’s content and structure, but after they have been cloned the current materials become effectively divorced from the past. So, not only do past students lose access to updates, they also lose their avenue to discuss any issues with the materials they have.

If we view university units only through the lens of delivered learning experiences, this separation of current and past offerings seems reasonable. The past class is over. If we view them through the lens that a subject is also a digital product, however, it is more problematic. How, for instance, can a past student raise or discuss a

bug in the product, which might or might not persist in the latest version?

Digital repository systems, such as openEquella (Aperio Foundation 2017) are a partial attempt to address the problem of maintaining digital materials through time, but come with the significant limitations that they rely on extensive metadata collection, they operate at the per-item level rather than describing a deployable unit design, and that as an institutional repository although they may contain the content item they do not represent how the community maintains and updates that item.

As I write this paper, I have just raised a bug, number #12420, against the compiler for the Scala programming language. This particular bug relates to how Scala 2.13.6 interacts with modules written in Scala 3.0.0. Before posting the bug, I discussed it on Scala's *gitter* channel. (Gitter is a text-based chat site associated with open source projects on GitHub). I was not prevented by an enrolment paywall, nor by the fact that Scala 3 has released and Scala 2 is last year's model. Within hours of my having posted the issue, another member of the Scala community had refined the problem I had identified, giving the Scala development team a simpler way to reproduce the bug. That bug has since been fixed and scheduled for release in version 2.13.7. After it is released, my Scala projects will pick up the fix just by updating one line of a configuration file to say they should use version 2.13.7 instead of version 2.13.6. I will have benefited by receiving the software fix and the Scala compiler will have benefited (in a small way) from two members of the community reporting and refining a bug.

The aspect that I wish to draw attention to here is not simply that openness brings mutual advantages, but that there is a level of technical sophistication in how open source software can be published, discussed, maintained, compiled, and released, that is still missing from open educational content publishing. A question I would like to pose, then is "What if content production were more like software production?" This is not a new question, but one the software development community has also considered in how it manages software documentation and project homepages.

Static Publishing and the JamStack

In 2008, Tom Preston-Werner, the co-founder of GitHub, developed a personal static blog generator, Jekyll (Preston-Werner and contributors, 2008). A key observation in its development was that popular content-management systems such as WordPress can have security flaws and can become targets for hacking. This continues to be true. For example, Trunde and Wiepl (2015) provide an analysis of 199 publicly disclosed SQL injection exploits that WordPress has encountered. Whereas, if a personal blog site is just a collection of HTML, JavaScript, and CSS files, then the blog author as an individual might have no server to hack.

Alongside Jekyll, GitHub introduced GitHub Pages – a feature whereby static files or a Jekyll blog could be committed to the *gh-pages* branch of a GitHub repository and would be published automatically by GitHub as a website. Competing tools for static site generation, such as Hugo (Pedersen et al., 2013), followed in subsequent years.

As GitHub is a key site for open source software projects, this quickly became a popular mechanism for those software projects to publish their documentation. Subsequently, a number of static *documentation* site generators have been developed that are specialised to this problem, such as ReadTheDocs (Holscher et al., 2010) and Facebook's Docusaurus (Docusaurus team, 2017).

More recently, static site generators have begun to take advantage of modern front-end web development frameworks, so that although the site is "static" in that it is served as a simple collection of HTML, JavaScript, and CSS files, when loaded and executed within a web browser it can include extensive interactive behaviour, including being able to make calls to online services. For example, Gatsby (Matthews et al., 2015) was developed as a static site generator based on the React front end system (Walke, 2013) that is commonly used to develop the front end of many web applications. Gatsby now lists 2,600 plugins to extend the functionality that the "static" sites it generates can offer.

As with many trends in software design, eventually someone creates some jargon to name them. In this case, these systems have begun to be nicknamed the *JamStack* (Bilman et al., 2016), roughly standing for the components commonly used in their creation: JavaScript, APIs, and markup. The *JamStack*, then, has evolved into a strategy for individuals to publish material to the web that can become highly interactive and make use of online services. Because JamStack sites tend to be published to open source software repositories, such as

GitHub and GitLab, they also gain the collaboration and community facilities that open source software development has long enjoyed. This includes public issue trackers for reporting of errors, public tracking of updates, public content delivery networks that ensure fast delivery and defend against denial of service attacks, and facilities for other members of the community to contributing fixes (for instance via pull requests).

These contrast well against our three usability issues about unit sites in Figure 1:

- Access to the materials is persistent, because the sites are open. Consumers of these sites, whether past, present, or potential students, always see the most up-to-date version of the materials.
- Although students would not usually need to actively retain content, because it remains open, past versions of the site are archived and accessible in its version control repository in a buildable form. (By downloading or cloning the *gh-pages* branch of a GitHub Pages site, it is sometimes also possible to take a working copy of the compiled site.)
- As the site is based within an open source software repository, its monitoring (for bug reports and updates) can be continuous. It contains a complete version history and it is simple to link to and track specific issues and updates within the site's history.

It is worth noting at this point that the concept of a code-like or mark-up-based production system for educational or academic materials has a long tradition. From academic papers to exam papers, one of the most common tools in static content production across scientific and mathematical fields has been LaTeX (Lamport, 1985). More recently, many software developers, academics and other presenters have used Markdown (Gruber, 2004) to produce documents and slides. The concept of “literate programming” (Knuth, 1984), which combines markup-based content with programmed data visualisation, also has a long history and remains popular today through systems such as Jupyter notebooks. So, the involvement of *some* programming-like code in content production is already well accepted.

A Step in Progress Towards Open Teaching on the JamStack

In previous work, I developed a number of interactive course sites that use a front-end framework to allow us to build interactive models into lecture slides, tutorials, and challenge environments (Billingsley, 2020; Billingsley & Vitale, 2021). Each site works as a *Single Page App* (SPA), taking advantage of the capabilities of modern browsers. They were deployed to GitHub pages and in many regards take do advantage of this JamStack agenda.

However, the sites those papers described were intentionally complex. At that stage, the focus had been on including a large number of highly interactive learning environments and simulations for students, and those simulations require significant programming skill and effort to produce. These were not sites that most academics would be willing to write.

The question, then, arises: What could be done for more regular courses – those that predominantly consist of slides, videos, tutorial instructions, and links to external repositories? How can the JamStack democratise public course production, in a similar manner to how static site generators democratised blog and website hosting to those who did not wish to host and maintain their own WordPress server?

In 2021, as described earlier, my Scala course needed updating to Scala version 3. Although as an *educational experience*, my course contains many programming exercises and a great deal of active learning, all those exercises are already externally hosted within their own GitHub repositories. A public *course site* for it only needs to hold the slides, video, written content, and links to those externally hosted tutorials. So, as a course site, it is similar in complexity to many other university subjects, and became a suitable testing ground to begin writing a course-oriented static site system: Doctacular.

Doctacular produces sites that are compiled to HTML and JavaScript files and can be hosted within a GitHub Pages site. This is similar to static blog generators such as Jekyll and Hugo, and static documentation site generators such as ReadTheDocs and Docusaurus. Like Gatsby, Doctacular is implemented on top of a rich front-end framework (Gatsby is built on React; Doctacular is built on Veautiful), allowing sites to include as many (or as few) interactive components as they wish. Like Gatsby, the site is produced as a Single Page App (SPA). This is a technical term that means that navigating within the generated course site does not cause the browser to attempt to load a new page from a server, but instead updates the displayed content dynamically.

A Doctacular site is defined by a human-readable top-level script that:

- Adds content to the site,
- Sets the table of contents,
- Optionally modifies any styling for the site, and
- Instructs the site to load itself into the page.

The steps of registering content and setting the table of contents can be combined, leading to a concise top-level script that is often little more than its table of contents. For instance:

```
site.toc = site.Toc(
  "Home" -> site.HomeRoute,
  "Getting started" -> site.addPage("getting-started", gettingStarted)
  // etc
)
site.home = () => site.renderPage(frontPage)
site.attachTo(document.getElementById("render-here"))
```

Within each entry, we need to know the title to display in the table of contents (e.g., “Getting started”), the path to show in the browser’s URL bar (e.g., “getting-started”), and the content to display (e.g., the content declared in another file as `gettingStarted`).

As a *course-oriented* static site generator, Doctacular understands that there can be alternative forms of the same content: for instance, a slide deck may be shown in notes form, or full-screen, or the video that was produced from it may be played. Again, this is done in a scripting style of coding, often within in the table of contents. Figure 2 shows an excerpt of the table of contents code for the Scala course.

```
site.toc = site.Toc(
  "Home" -> site.HomeRoute,

  "1. Imperative programming" -> site.Toc(
    "Intro" -> site.addPage("imperative", imperative.imperativeIntro),
    "Intro to Scala syntax" -> site.add("introScala",
      Alternative("Slide deck", Deck(() => imperative.introScala)),
      Alternative("Watch the video", Video(() => PlayableVideo(Echo360Video("92548534-cc2c-4661-99d4-f2dfc1e26309")))),
    ),
    "OO in Scala" -> site.add("objectOrientedScala",
      Alternative("Slide deck", Deck(() => imperative.scalaOO)),
      Alternative("Watch the video", Video(() => PlayableVideo(Echo360Video("302bd7dc-7a82-4d9a-a702-6efb5e94bb41")))),
    ),
    "Practical: Set up" -> site.addPage("tutorial-0", imperative.setupTutorial),
    "Practical: First Steps in Scala" -> site.addPage("tutorial-1", imperative.tutorial)
  ),
  "2. Functional programming" -> site.Toc(
    "Intro" -> site.addPage("functional", functional.functionalIntro),
```

Figure 2. An excerpt from the table of contents script for a live site

Slide decks are also defined in code, but hopefully only at the level of complexity of scripting. Many decks in the Scala course are simply a sequence of Markdown slides. For instance, in Figure 2’s table of contents, `imperative.scalaOO` refers to the deck called as `scalaOO` in the package (directory) called `imperative`. An excerpt of the code for that deck is shown in Figure 3, showing how we declare the size of the deck we want (1920 pixels by 1080 pixels) and then successively add slides to it as Markdown strings. However, we can also add more complex slides and interactivity if we wish.

```

val scala00 = DeckBuilder(1920, 1080)
  .markdownSlide(
    """
    |# Intro to Object Orientation in Scala
    """).stripMargin).withClass("center middle")
  .markdownSlide("""
  |
  |## Object-Oriented
  |
  |Scala is sometimes described as a fusion of object-oriented programming* and functional programming*.
  |
  |If you're coming from a Java background, you'll probably already be familiar with OO. But there are some
  |extra features that Scala has.
  |
  |So, let's
  |
  |* give a quick recap of what OO is, using a (slightly tired) analogy,
  |* and start to show you some of Scala's extra features
  |
  |""").stripMargin)
  .markdownSlide("""

```

Figure 3. An excerpt of a slide deck defined mostly as Markdown slides

In Figure 4, we can see how this renders in the live site. The table of contents in the left sidebar allows navigation between the different topics as they were laid out in Figure 2. The deck we are viewing (from Figure 3) is initially laid out in a notes format, with each slide rendered underneath each other for easy scrolling through the deck. Navigation at the top of the screen allows it to be played full-screen or navigating to the recorded video.

In both the notes and full-screen views, the deck auto-scales depending on the browser window's current size using a CSS scale effect. This allows content authors simply to write slides as if they have fixed dimensions (e.g., 1920 by 1080) regardless of what size they are really displayed at.

The screenshot shows a live site interface. On the left is a dark sidebar with a table of contents. The main content area shows two slides: 'Intro to Object Orientation in Scala' and 'Object-Oriented'. Navigation buttons 'Play this deck fullscreen' and 'Watch the video' are at the top right.

Figure 4. The deck rendered in notes format in the live site. The table of contents shows on the left and navigation between alternative views of this entry shows above the content

Doctacular is implemented as a Scala library, and the code in Figures 2 and 3 is (simple) Scala code using that library. The purpose of its inclusion in this paper is not, however, to give an in-depth description of Doctacular yet. It is only to illustrate to the higher education community its direction of travel and give a preview of its progress.

Academic Concerns in Copyright and Intellectual Property

Discussions of open publishing mechanisms are often expected to discuss the question of intellectual property – and this was recommended during the review process for this paper.

Technically, this is an orthogonal issue: a technique or mechanism for publishing content does not determine what licence it must be published under. However, I would argue that open publication using techniques from open source software publication can be helpful with this issue too.

Within universities, policies around intellectual property rights in content can be a complex story. For example, different institutions take different policy approaches at different times. Kelley et al. (2002), Kranch (2008), Hawkrigde et al. (2010), and Pila (2010), among others, have explored this space but it continues to evolve.

As I understand it, copyright ownership of unpublished materials within institutions might also become unclear (at least to the academics who write and use them) because academics habitually reuse materials that they have permission to reuse without transferring copyright in those materials. For instance, hypothetically consider the image in Figure 5, which I will say is an image, included in one of my courses, which I drew. I did not draw it at my current employer – at the time I drew it I was employed by a research institution but also teaching as an adjunct at different university. Meanwhile, the figure in the image has some similarity to *Marzy the Martian*, a character painted by my mother for a children’s story before I was born, and it is not the first time I have drawn the character or the scene. Without details that I have omitted on what agreements were or were not made and with whom (including my past employers), would it be easy to determine the copyright ownership of that image, sitting in the middle of one of my teaching decks? If many of an academic’s materials are entangled with their personal employment history, as they have edited them under multiple employers and policy regimes, perhaps in multiple countries, could their institution be sure of what its own rights in the materials are if they leave?

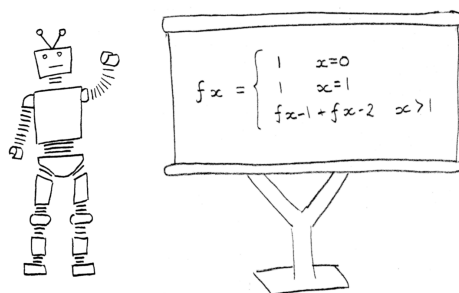


Figure 5. A simple drawing I drew whose copyright hypothetically might be harder to know

So, there may be an internal utility to institutions in openly licensing their content, of helping clarify their own use of content. A hypothetical academic moving from institution to institution during their career might create and edit the materials they use under multiple employers, potentially in multiple countries, accruing a complex history of copyright that may be difficult to know – but if it is openly licensed, every party (regardless of ownership) might be more assured of how it is intended to be used. Open source software projects often use a Contributor Licence Agreement, rather than a Copyright Transfer Agreement, to ensure that the project acquires from authors the rights it needs to operate the project and distribute the software.

The open source software community is also accustomed to complex forms of reuse and has developed tooling to support some of the licensing implications. For example, reuse can become transitive: one project uses a library from another project, which in turn uses a library from a third project – each of which may be under different licences. Gradle and the Scala Build Tool (SBT) are two popular build automation tools that have plugins available that can produce a report of the different licenses used in your project. Even though there may be many libraries with many licences, it becomes a one-line command to find out what they are.

In JamStack-style publishing, because sites are fundamentally programs, these more complex reuse relationships become possible in teaching sites and become feasible to track. In previous work (Billingsley 2020) I demonstrated one course site importing a circuit diagram and simulation model from another course site. This is a case of a Scala program importing a Scala library, so it would be feasible to run SBT's licence report plugin.

Deployment and Integration with the Learning Management System

Whatever language a JamStack site is written in, the site is normally compiled and published as a collection of HTML, JavaScript, and CSS files. As such, these sites can be “serverless” – that is, they can be published to any institutional or cloud web server, rather than relying their own specific server being maintained.

The sites that I have produced thus far are published personally to GitHub Pages, but for institutional teaching purposes are also published to an institutional web server. Effectively, this gives the advantage of having development and production sites. When I make alterations to the site code, the site is automatically rebuilt and republished on GitHub Pages. After checking I'm happy with my changes, I can then republish them to the institution's deployed site so they become visible to my students.

In these examples so far, the integration with the Learning Management System is just a set of links – the materials act somewhat like an open source textbook, but they do not directly attempt to make data connections. Some analytics are collected by university systems even in this scenario. The LMS records whenever a student follows a link from the LMS to the materials and the university's video hosting system record tracks whenever a video embedded in the course site is played. The published site itself, however, stores no data. This produces an interesting separation of concerns, whereby the content site is responsible for interactivity but does not store personally identifiable data, whereas the LMS knows and tracks who you are.

However, it is also possible for a JamStack site to create more complex relationships on the data side. For instance, sites may choose to store data locally within the user's browser. Or, they may choose to connect to services – Gatsby's plugins include analytics, e-commerce, and data source integrations.

In future work, I am exploring optional connections to data services for students and how those can change site behavior automatically. For example, we may wish course sites to behave differently depending on whether you access them via an institutional LMS and are therefore a currently enrolled student, or whether you access them anonymously as a member of the public.

Conclusion

The JamStack is an approach to website development that evolved in the open source software development community. It has become particularly popular for open source documentation sites and bears similarities with coding-like ways of developing content that are already established in academia, such as LaTeX. Because sites are written in a code-like manner (either through markup languages such as Markdown or through simple code), sophisticated forms of reuse and collaboration become available to them. As the browser has become a more capable and flexible platform for software, the capabilities of JamStack sites have expanded. As such, it appears to be a fruitful direction for the development of open course publishing.

A key aspect in the development of JamStack publishing systems is the developer experience. These systems gain much of their power and convenience from the fact that sites are written in a code-like manner, but the authors of teaching sites might not wish to see themselves as programmers. The level of coding complexity needed to produce or maintain a course site needs to be kept low.

Doctacular is a framework for generating interactive “static” course sites, written as a Scala library. It is a work in progress and currently under-documented, so I do not wish to make many definite conclusions about it in this paper. The presentation to accompany this paper is effectively an early demo of how it works. Later papers will describe the system in more technical detail. However, two courses have now been deployed with it. The Scala course went live in March 2021, with much of the content being written (updated to Scala 3) as I taught the unit. Our Computational Thinking course (Billingsley & Vitale 2021) was also “refactored” (altered) to use Doctacular in 2021. This allowed me to test that more complex sites that include simulations and programmable games within their decks and pages, can also operate well within its design.

References

- Aperio Foundation (2017). *openEQUELLA*. [Computer Software]. The Aperio Foundation. <https://www.apereo.org/projects/openequella>.
- Billingsley, W., Ngu, B., Phan, H., Gromik, N. & Kwan, P. (2016). Using a video based critique process to support studio pedagogies in distance education. In S. Barker, S. Dawson, A. Pardo, & C. Colvin (Eds.), *Show Me The Learning. Proceedings ASCILITE 2016 Adelaide* (pp. 43-48)
- Billingsley, W. (2020). Revisiting the Intelligent Book: Towards Seamless Intelligent Content and Continuously Deployed Courses. In S. Gregory, S. Warburton, & M. Parkes (Eds.), *ASCILITE's First Virtual Conference. Proceedings ASCILITE 2020 in Armidale* (pp. 230–240). <https://doi.org/10.14742/ascilite2020.0144>
- Billingsley, W., & Vitale, J. 2021. An Accelerated CS0 for Online Mature-Age Part-Time Students. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE '21)* (pp. 526–532). Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/3430665.3456361>
- Billmann, M., & contributors. (2016). *Jamstack: For Fast and Secure Sites*. Netlify. <https://jamstack.org/>.
- Cerf, V. G. (2011). Avoiding “bit rot”: Long-term preservation of digital information [point of view]. *Proceedings of the IEEE*, 99(6), 915-916.
- Cordes, D., & Brown, M. (1991). *The Literate-Programming Paradigm*.
- Coughlan, T., Pitt, R., & McAndrew, P. (2013, April). Building open bridges: collaborative remixing and reuse of open educational resources across organisations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)* (pp. 991-1000). Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/2470654.2466127>
- DocuSaurus team. (2017). *DocuSaurus: Easy to Maintain Open Source Documentation Websites*. [Computer Software] Facebook, Inc, Menlo Park, CA. <https://docusaurus.io>.
- Galloway, P. (2004). Preservation of digital objects. *Annual Review of Information Science and Technology (ARIST)*, 38, 549-90.
- Hawkridge, D., Armellini, A., Nikoi, S., Rowlett, T., & Witthaus, G. (2010). Curriculum, intellectual property rights and open educational resources in British universities—and beyond. *Journal of Computing in Higher Education*, 22(3), 162-176
- Hilton III, J. L., Robinson, T. J., Wiley, D., & Ackerman, J. D. (2014). Cost-savings achieved in two semesters through the adoption of open educational resources. *International Review of Research in Open and Distributed Learning*, 15(2), 67-84.
- Holscher, E., Johnson, A., Kaufmann, M., Fischer, D., & Gallegos, S. (2010). *Read the Docs: Create, host, and browse documentation*. [Computer Software] Read the Docs, Inc, Oregon. <https://readthedocs.org/>.
- Gruber, J. (2004). *Markdown* [Computer Software] <https://daringfireball.net/projects/markdown/>.
- Kelley, K. B., Bonner, K., McMichael, J. S., & Pomea, N. (2002). Intellectual property, ownership and digital course materials: A study of intellectual property policies at two and four year colleges and universities. *portal: Libraries and the Academy*, 2(2), 255-266
- Knuth, D. E. (1984). *Literate programming*. *The Computer Journal*, 27(2), 97-111
- Kranch, D. A. (2008). Who owns online course intellectual property? *Quarterly Review of Distance Education*, 9(4), 349-356,445
- Lamport, L. (1985). *LaTeX – A Document Preparation System – User’s Guide and Reference Manual*. pub-AW.
- Luo, T., Hostetler, K., Freeman, C., & Stefaniak, J. (2020). The power of open: benefits, barriers, and strategies for integration of open educational resources. *Open Learning: The Journal of Open, Distance and e-Learning*, 35(2), 140-158.
- Matthews, B., Shaon, A., Bicarregui, J., & Jones, C. (2010). *A Framework for Software Preservation*. *The International Journal of Digital Curation*, 5(1), 91-105.
- Matthews, K., Bhagwat, S., & contributors. (2015). *Gatsby: One front end to rule them all*. Gatsby, Inc, San Jose, CA. [Computer Software] <https://www.gatsbyjs.com/>.
- McGowan, V. (2020). Institution initiatives and support related to faculty development of open educational resources and alternative textbooks, *Open Learning: The Journal of Open, Distance and e-Learning*, 35(1), 24-45. <https://doi.org/10.1080/02680513.2018.1562328>
- Mishra, S. (2017). Open educational resources: Removing barriers from within. *Distance education*, 38(3), 369-380.
- Pedersen, B., Francia, S., & contributors. (2013). *Hugo: The World’s Fastest Framework for Building Websites*. [Computer Software] <https://gohugo.io/>.
- Pila, J. (2010). Who owns the intellectual property rights in academic work? *European Intellectual Property Review*, 2010, 609-613.

- Pitt, R., Jordan, K., de los Arcos, B., Farrow, R., & Weller, M. (2020). Supporting open educational practices through open textbooks. *Distance Education*, 41(2), 303-318.
- Preston-Werner, T. & contributors. (2008). *Jekyll: Simple blog-aware static sites*. [Computer Software] <https://jekyllrb.com>.
- Trunde, H., & Weippl, E. (2015). WordPress security: an analysis based on publicly available exploits. In *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services (iiWAS '15)* (article 81, pp. 1-7). Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/2837185.2837195>
- van der Hoeven, J. R., van Diessen, R. J., & van der MEER, K. (2005). Development of a universal virtual computer (UVC) for long-term preservation of digital objects. *Journal of Information Science*, 31(3), 196-208.
- Walke, J. (2013). *React.js* [Computer Software], Menlo Park, California: Facebook. <https://reactjs.org/>.

Billingsley, William. (2021). On the need for open teaching on the JamStack. In Gregory, S., Warburton, S., & Schier, M. (Eds.), <i>Back to the Future – ASCILITE '21. Proceedings ASCILITE 2021 in Armidale</i> (pp. 363–372). https://doi.org/10.14742/ascilite2021.0152

Note: All published papers are refereed, having undergone a double-blind peer-review process. The author(s) assign a Creative Commons by attribution licence enabling others to distribute, remix, tweak, and build upon their work, even commercially, as long as credit is given to the author(s) for the original creation.

© Billingsley, William. 2021