Open Source Software in Libraries

Open Source Software in Libraries

JOY PERRIN AND CHRISTOPHER STARCHER

Contents

Preface 1
Joy Perrin

RAIDER Publishing 3

1. Introduction 5
2. Software 21
3. Contributing to Open-Source Projects 58

Appendix 67

JOY PERRIN

The purpose of this book is to provide a short and quick introduction to open source software for students of library science. While the book is focused on open-source software in libraries, readers will get an overview of different kinds of technology and their purpose in a library and learn about the open-source equivalents.

Librarianship and the open-source movement share goals of making information open and accessible. This book should help students and practitioners get a better sense of what open source software is, and how it can be used in a library effectively. They will also learn how open-source software is probably already being used in their library by getting a review of the open-source software that is used in common technology.

The authors hope that this book will help increase the knowledge of the open-source movement among librarians and increase the quality of discussion about the movement's place in library missions and goals.

Joy M. Perrin is a Digital Initiatives Librarian at Texas Tech University with over 19 years of library experience at many different levels within the organization. She has held technical service positions, IT positions, and digitization positions. Throughout her career, she has utilized open-source software to meet the needs of patrons and to get her work done faster.

Christopher Starcher is a Digital Systems Librarian at Texas Tech University. He started his career as a music cataloger, moved to become a digital services librarian, and taught himself software development and library system management. Christopher has contributed to open-source software development projects and implemented open-source software to meet patron needs.

Within each chapter, there are stated learning objectives, focus questions for students to consider as they read, and at the end of every section is a reflection exercise to help students use what they have learned. Chapters end with Review questions, chapter summary, links to external resources, and a self-assessment for students to measure their own understanding of the material.

Chapter 1 is an introduction to the concept of open-source software and technology in libraries and how they interact.

Chapter 2 is an overview of the different types of open source software and how each could be implemented in a library.

Chapter 3 is a review of the various ways that librarians can engage with the open-source community at all levels of experience.

RAIDER Publishing

This book is an open-access publication under RAIDER Publishing, a publishing program started by the Texas Tech Library.

This book has not been peer-reviewed or edited.

If you would like to help make this text better and support the publishing initiative at the library, please consider giving to the TTU Library and requesting the money be spent on RAIDER Publishing.

Peer review costs: $500 ($250 per peer-reviewer) + Copy Editing and clean up: $1,500 = Total Cost for book $2,000, which is much lower than industry standard.

Every little bit helps.

1. Introduction

Learning Objectives

- To understand the role of technology in libraries
- To understand the background and history of Open Source Technology
- To understand the unique role of Open Source Technology in libraries

Introduction

Technology touches almost every part of a patron's experience of a library. The simple act of finding resources is just the surface level activity above a vast network of technology both library-specific and general. Understanding how the various technologies converge to the library and its services and how those technologies and services are affected by Open Source Software will greatly help librarians troubleshoot problems, plan for the future, and serve their patrons.

Focus Questions

1. As you read the following chapter, think about what technology you have taken for granted.
2. In what ways does the Open Source movement and library science goals match? In what ways do they resemble each other?
3. How successful has open-source software been for libraries? In what ways does it succeed and in what ways does it still need to develop?

Section 1: Technology in Libraries

Section Summary

This section talks about the history of technology in libraries and how that history informs the current state of libraries in general.

Library Technology History

Libraries, from their earliest inceptions, have always used the technology available to get information to their patrons. Sometimes that technology took the form of processes and organizations, but starting in the 1960s, the concept of computers as a part of library technology emerged [1]. In 1965, the Library of Congress released a report that "concluded that **automation** in bibliographic processing, catalog searching, and document retrieval was technically and economically feasible"[2]. What was needed still were systems designed for libraries, and for libraries to have their data in formats easily machine-readable. From this need emerged the **Machine Readable Cataloging (or MARC)** standard. The format was designed to not only be read by a computer but also shareable so that many libraries could get the benefit of one library's cataloging work. The creation of a sharable standard meant that a central repository of cataloging could be maintained, and that made the **Online Computer Library Center (OCLC)**, a non-profit organization whose mission is to make the technological cost of libraries lower, possible. As technology advanced, libraries were able to connect their resources using a network, and in 1971 the Alden Library at Ohio University became the first to do online cataloging [3].

This is a theme that will reoccur. Technology not only lets new and interesting things happen but in the case of software and data, it allows cooperation. It allows many organizations to benefit from the work of a few.

This caused librarians to realize the potential of **digitization**, which is the act of converting a physical item into an electronic or digital item. In the 1980s, the library of congress started their

1. https://www.ifla.org/files/assets/information-technology/publications/40-years-of-its.pdf
2. https://web.archive.org/web/20140305103046/https://blogs.loc.gov/loc/2014/01/a-half-century-of-library-computing/
3. https://web.archive.org/web/20121004135231/http://www.oclc.org/us/en/about/history/beginning.htm

American Memory pilot project, with the goal of making "5 million items accessible electronically to the nation by the year 2000" [4]. If cataloging could be made more efficient with technology and shared resources, so could books and other items. Also around this time libraries developed the **Online Public Access Catalog (OPAC)**, which allowed patrons to search the catalog easier and faster using text-based searching.

In the '90s, the idea of a library website as a portal to the library's services and resources was starting to take form. CD-ROMS were being sold with **electronic resources**, and those developed into the databases that libraries subscribe to today.

The 2000s have been marked by rapid growth in internet technologies, computer languages, and other tools that have helped libraries change rapidly. Caught in the middle of this change are library staff who have to balance traditional tasks and goals with a rapidly changing information environment.

Most modern libraries have a strong foundation of library technology. Most of the technologies that the library science world focused on for the past 60 years are now so commonplace as to be nearly invisible to patrons. Computers are expected. On those computers, patrons expect the internet. They expect to be able to search and find resources from around the world easily. This seamless experience for the patron comes at a technological cost for the library. Efforts must be made to break down barriers and create seamless experiences. Library employees are hired to work with vendors or to maintain servers and software, prevent downtime, and upgrade as new and better technologies are developed. The traditional focus of the librarian as an organizer of information has shifted to include organizer and maintainer of

4. https://web.archive.org/web/20140305103046/https://blogs.loc.gov/loc/2014/01/a-half-century-of-library-computing/

information systems. Many new job postings require applicants to stay abreast of future trends, and many are looking for librarians who can be relied on to do the work those future trends require.

The challenge to new librarians is quickly developing the technology competencies that they will need in their future jobs to be flexible [5].

In addition to that, Librarians are at the forefront of spreading digital fluency, fighting for user-centric design, fighting for patron privacy in a world of large companies that sell user's data. Librarians are at the crossroads where technology and evolving information landscapes converge. It is vital that librarians, in whatever capacity they can, to be well versed in how the technology works and interacts with both patrons and library and information policy. Practice and experimentation are two great ways to become familiar with these concepts. The Open Source Movement provides a backdrop on which librarian practice and experimentation can happen.

Reflection Exercise

Think back on the last time you visited a library in person and the technology that was present that you may have taken for granted. Did you walk through an alarm system that would go off if someone stole a book? Did the circulation desk have barcode scanners? Does the library have a self-checkout machine? How many other patrons were there using the computers for

5. https://www.sciencedirect.com/sdfe/pdf/download/eid/1-s2.0-S0099133318301848/first-page-pdf

something other than looking up books in a catalog? Were other patrons charging their phones? What technology education opportunities did the library provide? What services and events would be impossible if the electricity went out because of an emergency?

Section 2: The Open Source Movement

Section Summary

In this section, the core aspects of what open source software is and why it exists will be discussed.

Open vs Free

Before talking about the movement it is important to understand the difference between the words “open” and “free”. They are not interchangeable when it comes to **software**. Open means that people are allowed to see the **source code** and edit it, but it doesn’t necessarily mean that the software is free. A good example of this is the operating system Linux. While the software is open, and you can get a version of it for free, some companies have developed their own versions of the software that you have to pay to get. Free software, on the other hand, might be available at no cost but the code isn’t open so it can’t be edited and re-used. Some people talk about free software and what they mean is software that is both open and free from cost to use. When software is not free from costs, and it is not open, it is called **proprietary software**. Companies are usually disinclined to allow other people to see their source code, let alone edit it.

Some people feel that software should be free speech and they feel locking it down or making it proprietary is ethically wrong. Those people belong to the **Free Software Movement** which started in the early '80s. The free here stands for both free from costs and open to be edited and remixed. The **Open Source Movement** was a branch of the Free Software Movement and consisted of people who believed that proprietary software was fine but still wanted to advocate for open options.

The Open Source Movement is about creating software that allows anyone to edit, remix, or reuse all or part of the software. The movement has more to do with licensing, permissions, and organization of **development** than it does the actual programming. In much the same way as **Creative Commons' licensing** allows creators and authors to expand on copyright to allow for different uses, the Open Source Movement has recommended licensing for software projects to allow people the legal permissions to edit software [6]. The licenses used are typically the **GNU General Public License (GNU GPL)**, but there are other free software licenses out there [7]. The foundation of the license is that the software is free to be used for any purpose, to be changed for your needs, the freedom to be shared, and the freedom to share your changes. Software created with Open Source or free licensing is known as **Open Source Software (OSS)**.

A lot of the benefits that libraries found for shared cataloging can also be found in OSS. A few developers can create software that the whole community can use, and if someone wants a new feature

6. https://www.gnu.org/licenses/licenses.html
7. https://www.gnu.org/licenses/licenses.html

they can program it and release it back to the community. Just like how OCLC came about to organize all this shared cataloging, organizations developed to organize shared OSS programming projects. **GitHub** is a very popular software organization platform that provides some features for free and some for a fee. GitHub is based on and uses the **Git** standard which is used for **version control** of source code. Git itself is a OSS. Because OSS software is open, it can usually be run for low cost or free, but this is not always the case. The openness of OSS software projects also means that there is potential for a diverse developer community which means many different developers can look at the same code and see different ways to fix problems. It also means there's the opportunity to be more flexible than other software because anyone can submit changes.

Wikipedia: An Example

The Open Source Movement, and the communities within in, are developing and figuring out how better to run projects and maintain documentation. A great example is Wikipedia. Wikipedia is an online encyclopedia that launched in 2001, and is run off of the software Mediawiki [8] which is an OSS, which means it was developed and is currently maintained by developers from around the world and from many different backgrounds. This has resulted in a well maintained and secure software that is used for projects even outside of Wikipedia. Mediawiki is a good example of how OSS can work well within an open-source **distributed development** environment. Some librarians have taken up Wikipedia as a tool to help people find reliable and open information and have created

8. https://www.mediawiki.org/wiki/MediaWiki

library programming to help librarians understand how to use it [9]. Librarians do this because patrons often go to Wikipedia first when formulating their search strategy when they are looking up terms to use in a library system search [10]. This is an example of how an OSS outside of libraries can affect librarians and library patron search behavior.

Reflection Exercise

Have you used Wikipedia in the last week? Were you aware that it was an open-source software project developed by both paid and volunteer developers? Do you have an android phone? Have you ever used the Firefox internet browser? Have you ever had a WordPress website or blog? The internet itself is based on open-source software like Apache, MySQL, PHP, and JAVA. These are all examples of open source software that you might have used without really thinking about where the software came from or whether you could access and edit the source code.

Section 3: Open Source Technology in

9. https://meta.wikimedia.org/wiki/The_Wikipedia_Library/1Lib1Ref
10. http://www.washington.edu/news/2010/03/18/uw-study-finds-wikipedia-first-stop-for-many-student-researchers/

Libraries

Section Summary

This section is an overview of open source technology in libraries and the potential benefits it has for librarians.

Open Source Technology in Libraries

"The principles and practices of open source software are very similar to the principles and practices of modern librarianship. Both value-free and equal access to data, information, and knowledge. Both value the peer review process. Both advocate open standards. Both strive to promote human [understanding] and to make our lives better. Both make efforts to improve society as a whole assuming the sum is greater than the parts."

– Eric Lease Morgan [11]

There is an abundance of open-source software for use in libraries. This ranges from common technology that everyone shares, to specific library technology developed for a specific purpose.

Libraries are moving towards open-source software for a variety of reasons [12]. Some libraries see open-source software to control the "speed and direction of their library migrations", or to have modern functionalities in systems that were more recently created with newer technologies in mind. Some libraries have moved to

11. http://infomotions.com/musings/biblioacid/
12. https://timreview.ca/article/177

open source software so that they could own and maintain the data of their systems. In proprietary systems, this data is sometimes owned by the company. Some libraries move to OSS because it is more flexible and changeable for their local developers to make it work the way they want. Other libraries have moved to OSS because of how closely OSS matches with the library profession in general like in the quote at the beginning of this section.

Other libraries specifically avoid OSS because they are concerned about legal issues or having no control about the external development of the software. Maybe they worry it's not as safe as proprietary software [13]. Or perhaps they are dubious of the idea that the software might be cheaper than proprietary software. Others avoid it because OSS software often doesn't have the professional look that proprietary software have, and that patrons are expecting. Other libraries don't like the do-it-yourself nature of OSS, and prefer to have a company to call for problems and feature requests.

There are levels of technology, as well as levels of technology adoption and there are benefits for librarians at all levels. The adoption of an open-source Integrated Library System (ILS) is a massive undertaking that may not fit all institutions. More approachable is the idea of providing open-source software to patrons on the computers they use. Even more approachable is the single librarian toying around with open source software to get experience and try new things without ever having to bring the software to the public or request software to be purchased.

Within the realm of public-facing software, there are also levels of involvement. A library can use open source software that functions in almost every way like proprietary software by hiring a company

13. https://timreview.ca/article/177

to run and maintain the software for them. Or, libraries can join **consortium** that run OSS as a part of the member benefits, so that the group gets the benefits of the software and the benefits of the built-in **community support**. On the other end of the spectrum, the library can hire staff to maintain the software and the servers **in-house**. The aspects that go into this choice will be discussed in Chapter 3: Moving to Open Source Software.

The Dilemma of Collective Action and the 2.5% Proposal

In February of 2017 John Wenzler wrote about the dilemma of collective action which is that for libraries to reap the benefits and savings of new technologies, it requires a level of coordination that is impossible no matter how obvious the benefits are [14]. It is difficult for libraries to argue that money or time should be diverted away from local needs, which hinders their abilities to work toward collective action. OSS, however, is an example of how the "dilemma of collective action" can be overcome, at least in specific circumstances. In a perfect world, libraries that could develop would group together to develop the perfect library software that other libraries would use. They would all choose one project and put all their efforts on it, make it work for the whole community, and make it as cheap and easy to run as possible. To support this, David Lewis wrote a response that suggests every academic library put 2.5% of their budget toward the development of open-source software to create a common infrastructure [15]. It remains to be seen if this approach is the best solution for libraries, but it does show that open source software still holds some potential to be a conduit for change in the library world.

14. https://crl.acrl.org/index.php/crl/article/view/16581
15. http://hdl.handle.net/1805/14063

Why Librarians Should play with Open Source Software

With most technologies librarians interact with, especially software, it can be like a black box. You provide some input or a command, and something happens inside the box and a result comes out. OSS, on the other hand, can be pulled apart and examined. Looked at through a microscope and studied. The algorithms at play can be inspected for bias, or you can simply try it out without having to contact an IT person. For library science students, open-source software provides a playground for getting experience in parts of systems that are normally locked away such as the back end of library systems.

Even if you never program a single line of code, knowing how a system works and how the development of OSS works, can better prepare you for writing documentation, coordinating with IT staff, and maybe running other kinds of projects.

When you look at a problem in your daily work, and you realize you might have a need for software, just knowing that there is free and open software you can dip into is an incredible boon. The library employee that can look at a problem and quickly download and run software to solve it is a valuable employee that can fill in technological gaps in a library workforce. When you see a metadata problem, you see a data problem, and you can pull out tools like OpenRefine (an open-source desktop application for data cleanup and transformation). You have a patron with a need to convert an audio file, but your library doesn't have the software. You know you can go get Audacity (which is a OSS platform for audio manipulation and editing). Maybe someone wants to have a knowledge base for their work, and you know how to run and maintain Mediawiki. Maybe your team wants to create a mind map but there's no budget for software, you can get Freemind installed on everyone's computer without a single penny. Maybe you have been put in charge of a large project for the first time, and you find GanttPV (an open-source project management tool). Maybe you need to create a website from scratch, and you know to look for Drupal (an open-source website management software). The examples could go on, but in general, if you find a piece of software you need you can usually find OSS versions to play with.

Build a Tool Box of Open Source Software

In my entire career, I've found multiple times where I was able to dip into OSS to complete a task quickly where another employee would have had to do it by hand. I once had to re-name hundreds of files, and I was able to download Ant Renamer and do the work in a few minutes instead of weeks. Other times I've had to edit metadata and I've used OpenRefine to do what MS Excel couldn't. I've used MediaWiki, and open-source server software to run projects and prototype ideas without having to spend a dime.

If anything, for me, OSS brings a certain freedom to experiment without having to worry about cost. If it's a software I can download to a computer, I can play with it.

I think it's important for every modern librarian to develop a toolbox of OSS that they are familiar with that they can bring to various library problems.

This book will introduce you to various OSS projects, explain how to manage the adoption of OSS in libraries, and talk about further contributing to the OSS community, and how to develop your own developer skills.

Reflection Exercise

Of the examples of OSS projects listed, did any of them pique your interest? Dive further by looking into the software that you thought was most interesting and look at the features available. Does knowing that these software projects support the open exchange of ideas make you feel differently about them? Do you think libraries should be using at least some OSS? What do you think are the barriers to libraries using OSS besides the ones listed in this section?

Review Focus Questions

1. As you read the following chapter, think about what technology you have taken for granted.

   Did you find yourself thinking more about the technology you use day to day while reading this chapter? Has any of the knowledge changed the way you feel about the software on your computer, your phone, or on the computers at your local library?

2. In what ways does the Open Source movement and library science goals match? In what ways do they resemble each other?

The quote at the beginning of section 3 does a great job of summarizing how OSS and libraries share common goals. Are there any other ways that they are similar? Do you think there are ways that each community could support the other?

3. How successful has open-source software been for libraries? In what ways does it succeed and in what ways does it still need to develop?

   This will be discussed more in the next chapter, so if you don't have a clear view of this yet that is ok. Maybe you understand that some OSS software is difficult to implement while others are easy. This question is one we will return to at the end of chapter 2.

Chapter Summary

In this chapter, we went through a short review of the history of technology in libraries so that we could gain context of how open-source software may fit in. It then discussed the Open Source Software Movement and the reasons and motivations for the movement to have started. It then talked about how Open Source Software has potential for libraries and librarians.

2. Software

Learning Objectives

- To be familiar with different kinds of software and the purposes behind them
- To understand how OSS has affected these kinds of software
- To have a basis for building a functional OSS toolbox to use

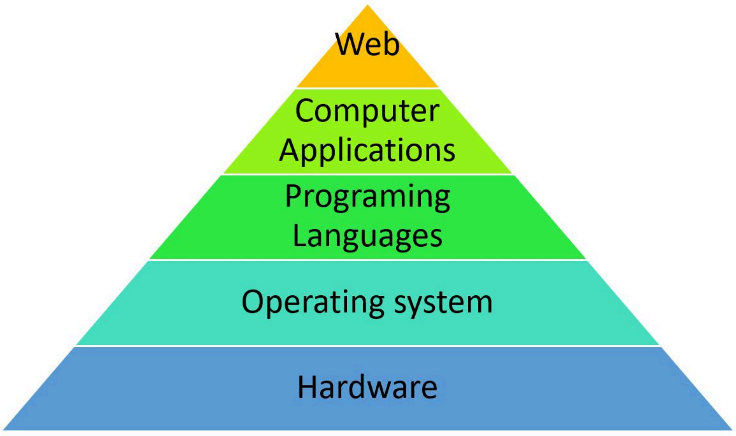
Introduction

In development, there is a thing called a "full stack developer". In theory, this is someone who understands and can work in the various levels of technology that build up our current software environment. There is some argument that the "full stack developer' is a myth because the software environment has gotten too complex to understand fully. There are so many different software languages to learn, and various software to link together.

The goal of this chapter is to help students understand the levels of technology and why they are there. Readers will develop a conversational understanding of the "stack" of software. If you're talking to a patron and they have a problem with the website, you may pick up on the fact that the problem is their browser. When someone is talking about a new technology, and they start going into details, you'll get at least an idea of what they're talking about or what level of the stack of software they are discussing. Even if you are never a part of a software development project, you may at some point find yourself troubleshooting a piece of software and the knowledge of the different levels of technology that might be going awry may help you in at least identifying the problem to tell your IT support. Or, you are accidentally the IT support for the day and you must fix the problem yourself. If you get into a job and see a technology opportunity, such as a maker space or a position of an integrated systems librarian, you'll be better prepared to take advantage.

The content of this chapter will help you develop that top-level understanding of the technologies. The learning activities at the end will guide you toward getting the first taste of hands on experience with the different levels.

Figure 1 shows the levels of technologies discussed in this chapter as a pyramid because they build on each other. The first section will talk about the foundation of the pyramid, and the last section will talk about the top.



Levels of technology

Figure 1 Levels of Technology

Focus Questions

1. How familiar are you with the software you interact with on a day to day basis? As you read through the chapter, think about the levels of software that build up to make even a simple application run.
2. What kind of software do you feel most comfortable with? Which kind feels the most confusing to you?
3. What comfortable steps could you take to make the software you are least comfortable with more familiar to you?

Section 1: Bottom of the Stack

Section Summary

The bottom of the stack is the stuff that literally must be in place for any of the rest of it to work. This section review hardware, operating systems and programming languages.

Hardware

All software is run on some form of **hardware**. The hardware is the individual components of a computer. Hardware is called that to distinguish it from software, which are the programs that run on a computer.

The hardware of a computer usually has the same components that do jobs. There is the case, which gives the components something to attach to. The brains of the computer (**Motherboard** and **CPU**), the **power supply**, the memory (**RAM**), and then additional attachments that do various tasks like the monitor that let you see a graphical representation of the computer's **output**. Computers can be built from individual pieces purchased separately or purchased as pre-made systems. They can be smaller or larger, but the point is they usually have all these parts in one form or another.

In the same way that software can be open, the designs for hardware can be open as well. **Open Source Hardware** is an example of how a design can be open with the licensing that allows all kinds of editing and reuse, but the item itself still costs money to build or have. Open Source Hardware is built on the same founding concepts of open source hardware that the design is licensed so that the design can be studied, modified, and distributed [1].

In addition to being open, the design must be accessible. Most open-source hardware designs seek to use non-specialized equipment, easily attainable parts, and instructions that are easy to follow. The point here is that for designs to be truly open source, they can't just be available. They must be functional for most

1. https://opensource.com/resources/what-open-hardware

people. There are open-source designs for bikes [2], open-source 3D printing equipment [3], and full open-source computers.

More popular than full computers are individual open-source hardware parts. The most popular and well known is the **Arduino**. The Arduino is a platform of both circuit board and software [4] that is a favorite of home hobbyists. Arduinos can be used to create things like garage door openers, talking clocks, thermostats, pollutant sensors, and a wide range of other projects [5].

The openness of the Arduino design means that multiple companies can make and sell Arduino devices and so there are lots of different versions. Consumers get the benefit of this competition by having lots of options for various projects, and a continuously developed hardware.

It might be impractical to use open-source computers in a library setting, but library makerspaces can make use of Arduinos and open source 3D printing equipment and designs. Libraries can provide workshops and materials to not only show people how Arduinos can be used, but also talk about the concept of open-licensed hardware and software. This is just one way for libraries, with their focus on openness and freedom of information, to use their makerspaces to support open-source hardware development and community.

2. http://www.xyzcargo.com/
3. http://www.appropedia.org/Open-source_3D-printable_optics_equipment
4. https://learn.sparkfun.com/tutorials/what-is-an-arduino
5. https://readwrite.com/2014/03/29/10-arduino-projects-microcontroller-electrical-engineering/

Operating System

Operating System (OS)

The shortened form of “Operating System” is often referred to as OS.

Because we are referring to Open Source Software as OSS, we’re not complicating things by adding OS as a shortening for Operating System. Just realize if you are in a situation and someone throws out the following phrase “UNIX is basically a simple OS, but you have to be a genius to understand the simplicity”, you will understand they are talking about an operating system.

By the way, that quote is from Dennis Ritchie, the guy who created one of the most influential programming languages of our time. He also co-created the operating system mentioned in the quote.

The **Operating System** is the foundation of how software is run. It is the thing that connects the physical computer (Hardware) and the program (Software) so that they work together. The operating system is a bit like the brainstem of the computer. In the same way that your brainstem controls the beating of your heart and the rate of your breathing, the Operating Systems control simple procedures like taking input from the keyboard or managing how data is treated.

A handful of operating systems dominate the computer landscape. **Windows** is the most popular operating system for desktop and laptop computers with 82.45% of the market[6]. **Mackintosh** (or Macs) are the next in line with 12.64% of the market. The most popular OSS Operating System is **Linux**, but it only holds 1.7% of devices worldwide. It is important to note the difference in scale here. Even though the number of devices that run Linux are low, it remains an influential operating system.

When we talk about Linux, we're not talking about specific software. Instead, we are talking about something called a **kernel**. The kernel is the core of the system and how it works, but it's more like a framework that other functionality can be built on. Lots of operating systems use the same kernel, and if that software is using the Linux kernel it's considered a Linux operating system, but it may have a different name. Red Hat Linux, Fedora, Mandrake Linux, Oracle Linux, and Ubuntu are all different Linux operating systems. With a proprietary Operating System like Windows, you only get the flavors of the software that the company wants you to have (Windows 7, Windows 10, Windows XP), but with OSS operating systems, there can be hundreds of different flavors for different purposes and people. Some Linux systems are better for desktop computers, and others are designed to be stable **server software** (more about that in Section 2).

There are free versions of Linux, but there are also pay for versions since the license allows for it to be used in both non-commercial and commercial distributions.

In addition to the different flavors of Linux operating system, there are different **distributions** of the software. The Linux operating system is kind of useless by itself. It needs other software

6. http://gs.statcounter.com/windows-version-market-share/desktop/worldwide/#monthly-201806-201806-map

to make it more useful for the common user[7]. Some common things to be included in a distribution, in addition to the Linux Kernel, are things like a desktop environment, an installer, an email client, web browser, or video and audio players.

For libraries, it is important to note that **Linux environments** are completely different than **Windows environments** and most **IT people** are familiar with Windows, but it takes special knowledge to be familiar with Linux. OSS often runs on Linux operating systems, so if your library wants to run OSS software it will need to find or develop Linux operating system experts. You can, through self-study and experimentation, become that expert, but it will take time and dedication. The learning activities at the end of the chapter will help you get started down that path.

Linux, despite its steep learning curve, still has some substantial benefits. Because it's not as popular as Windows, it is rarely targeted by malware. The kernel was also designed to be stable, so it's less likely to **crash**. Some people really enjoy running Linux on their personal computers and are willing to deal with the learning curve to get away from "control-freakish environments that Apple and Microsoft have increasingly foisted on users of personal computers"[8].

Programming Languages

Software can't exist without someone sitting down, designing it, and typing out the **code** to make the software work. Instead of

7. https://www.lifewire.com/basic-guide-linux-operating-system-2202786
8. https://www.wired.com/2016/01/i-moved-to-linux-and-its-even-better-than-i-expected/

having to learn **machine code**, programmers use a language that is easier for our brains to understand and that usually follows human logic patterns. These languages are generally referred to as **programming languages**.

Every other category in this chapter is usually dominated by proprietary software, but the programming language arena is dominated by open source software languages[9]. The reasons behind this have a lot to do with the value of OSS in general. Proprietary languages cost money and they're limited (but they can be useful in some circumstances). OSS programming languages change as needs change. Their openness makes them more accessible, especially for those trying to get experience in programming without having to spend money.

Not all programming languages are created equal. Since programming languages are created for different tasks and different people, they have different strengths. Languages are based on different kinds of programming logic, like **object-oriented programming** [10], **functional programming**, and **procedural programming**. It is not important now to understand the difference between these, but it is important to understand that different languages are designed to function in fundamentally different ways. Some languages are flexible enough to be used with multiple kinds of logic, and some require very rigid adherence to one or the other.

To illustrate this point, we will review two open-source programming languages in depth.

9. https://www.wired.com/2015/08/github-data-shows-changing-software-landscape/
10. https://searchmicroservices.techtarget.com/definition/object-oriented-programming-OOP

Machine Code

01101101 01100001 01100011 01101000 01101001 01101110 01100101 00100000 01100011 01101111 01100100 01100101

That list of zeros and ones is the words “machine code” written out in **binary**.

6d 61 63 68 69 6e 65 20 63 6f 64 65

That set of letters and numbers is the same words written in **hexadecimal (hex).**

These languages are programming languages that talk the language of computers and not people. The computer can take this input and respond directly to it.

Very few people can program in binary or hex because it is so far removed from the language and logic our brain uses.

Instead, most people use a programing language to write out statements in a logical format that we can understand, and then they **compile** it into machine code using another program.

Java

Java is the most popular programming language for OSS [11]. It can be considered to be a general-purpose language that is designed to run

11. https://www.wired.com/2015/08/github-data-shows-changing-software-landscape/

on any operating system and on any **computer architecture** (a term that is related to the computer hardware). Java started it's life as a proprietary programming language but was released under Open Source licensing. Java was created with very specific programming goals in mind [12]. It needed to be simple, and use object-oriented programming logic, so that people could use it quickly without in-depth programming or development knowledge. The object-oriented focus meant that programmers could focus in on creating bundles of code that did specific jobs, and then call on that code when needed instead of having to re-create it. A programmer can, for example, create a function to create a random number (or use one already out there), and instead of having to re-type the code, they can just reference the random number function. In coding, this function would be considered an object, which is where the object-oriented part of the name comes from.

Because of Java's popularity with OSS projects, and it's platform-independence, and it's low-level programming nature, it's a language that packs a lot of potential for the investment of time into it [13]. It is not the easiest language to learn, so if you choose to investigate it take it slow. If it doesn't work out, try out the next one.

Python

The Python programming language is another general-purpose programming language whose design was built around the idea of making it easy to read. This also means that it's easier to maintain because multiple programmers can look at the code and understand

12. https://www.oracle.com/technetwork/java/intro-141325.html
13. https://simpleprogrammer.com/best-way-learn-java/

it easily [14]. This is not always true of programming languages, and some (like PERL) are notoriously hard to read and understand even if you know the language. It is flexible enough to work in all the programming logics mentioned at the beginning of this section (so it can do Object-Oriented programming and the others).

Python is an organized logical and powerful language that has a lot of potential depth. It's clear readable syntax, quick progression, versatility, available open resources, and supportive community make it a great first language that can teach you a lot of things about programming[15]. It's also popular as an OSS programming language.

Libraries and Programming Languages

Even if you never end up writing a program that other people use, there are several good reasons to at least be casually familiar with programming languages. As patron needs change so do software requirements, so often libraries hire developers either full time or temporarily to do that development work. It helps to be able to understand what is possible and what is not possible in programming and to be able to speak some of their terminologies. A librarian with a little bit of programming knowledge but a lot of library domain knowledge working with a developer can make big waves in librarianship. A good example of this is Occam's Reader,

14. https://www.wefearchange.org/2010/06/import-this-and-zen-of-python.html
15. https://mikkegoes.com/5-reasons-why-python-is-a-great-first-programming-language/

a project created by an Interlibrary Loan Librarian and a Software Developer to lend eBooks across libraries [16].

On a more personal level, knowing just a little bit of programming can help you do small **scripting** tasks. Let's say you end up needing to convert thousands of images into PDF and you don't have the proprietary software to do it but you do know a little Python. You can borrow some code, understand it, and use it without having to create a whole piece of software to do the job [17]. There are many small tasks in libraries that people run into day to day that could benefit from some quick scripting. Examples include dealing with files and converting data. There are even cookbooks for different programming languages that allow you to borrow code to do specific tasks, so you don't have to create everything from scratch.

Just remember, you don't have to be a software developer to understand code and to use it. You don't have to create software to use scripting to do a quick task.

Reflection Exercise

Think back on the focus questions at the beginning of this chapter. Of the software that you felt least comfortable with, how much of it was at the bottom of the stack of software? Because this is the foundational stuff, most people move through life not having to work with it, so they don't have a reason to know it even

16. http://publiclibrariesonline.org/2016/05/occams-reader-interlibrary-e-book-loans/
17. https://stackoverflow.com/questions/27327513/create-pdf-from-a-list-of-images

exists. Before reading section one, did you feel that this is true of you? Did your opinion change or stay the same after reading section one?

Section 2: Middle of the Stack

Section Summary

More people are comfortable with the middle of the stack software. This is stuff like word editing documents, email managers, and music software. However, most people are less familiar with server-level software even though it's in the middle of the stack.

Computer Software

This is the area of OSS that most people feel most comfortable with. It's easy enough to download a piece of OSS software and mess around with it without having to worry about operating systems and programming languages. Most computer level OSS is compiled to run independently. You may have even already dabbled in computer level OSS without realizing it.

It is possible to have a full software suite of nothing but OSS. The only exception is open source Antivirus software. Any open-source antivirus software should be used with caution. In Table 1, common desktop application tasks are listed with popular OSS versions of the software. In many cases, there are more options than those listed, but this list is provided as a starting point.

Task	OSS	Notes
Office software	LibreOffice Apache Open Office	Word editing, spreadsheets, presentation software.
Photo and Image Editing Software	GIMP	Replaces software like Photoshop. Used to edit images and photos or create graphics.
Audio editing software [18]	Audacity Linux Multimedia Studio	Playback audio files edit various elements of the audio (change pitch etc). Save audio in different formats.
Desktop Publishing [19]	Scribus LaTex	Creating things like documents, calendars, various printouts. The Office software may also have desktop publishing software as part of it.
Computer Design Software (CAD) [20]	Blender 3D	Creating 3D designs.
Web Browsers [21]	Firefox Chromium (Chrome)	Browsing the internet.

18. https://beebom.com/best-audio-editing-software/

19. https://pagination.com/desktop-publishing-software/
20. https://blog.g2crowd.com/blog/cad/10-open-source-free-cad-tools-download/
21. https://www.makeuseof.com/tag/best-open-source-browsers/

Email Clients 22	Opera Mail Mozella Thunderbird	Used for checking email, sending email, and searching and storing email.
Mind Mapping Software 23	Freemind	Mind mapping software is often used to organize thoughts or work.
Project Management Software 24	GanttPV MyCollab- Requires Javaruntime, MySQL stack OrangeScrum Odoo]project-open[– Core is open source, but modules are not. LibrePlan ProjectLibre	Keep track of projects, tasks, time frames, and costs.
Online Cloud storage or file sharing 25	ownCloud NextCloud Seafile OnionShare Pydio Cells	A new category of OSS, because it's a new technology in general. The purpose of this kind of software is to give you "cloud storage" of your files, and ability to share them.

AntiVirus	ClamAV Armadito AntiVirus Moon Secure AV	Most of the Open Source AntiVirus software should be used with caution.

OSS desktop software does have its problems. Often, the software does not work the way you have learned these kinds of software work because it was created independently of the software you have experience with, so expect a learning curve. Not all OSS is well documented. Some software might be fine for your purposes, but if you have a problem you might have difficulty in finding a solution. OSS software is not always pretty. It may function, but it may not look as polished as proprietary software.

In general, start slow and try OpenOffice [26]. It's a great introduction to what OSS desktop applications can be like, and you likely have experience with other word editing and spreadsheet applications to compare it to.

22. https://www.techradar.com/news/the-best-free-email-client
23. https://business.tutsplus.com/articles/best-mind-mapping-software-tools--cms-29581
24. https://opensource.com/business/16/3/top-project-management-tools-2016
25. https://opensource.com/alternatives/dropbox
26. http://www.techsoupforlibraries.org/planning-for-success/innovation/free-and-open-source-software-in-libraries

Server Level Software

The internet is a series of connected wires, but those wires communicate and talk to each other because of server software. The internet will be dealt with in the next section. For now, lets focus on what a server is and why you should care about OSS server-level software.

What is a server?

A server is just a computer that is designed to serve data to another computer (hence the name). The computer has special software that allows it to talk to other computers using **protocols**. Any computer can be turned into a server by installing server software on it, but whatever content you make available will only be available if that computer is turned on. Computers that act as servers for websites (like Amazon, or Google) must be designed to run constantly and be resistant to problems.

Overall, servers are designed to be computers that communicate. The computer you're working on, if it has a web client, sends out a request to talk to the server that has the information it wants. The other server gets the request and, just like us, has to decide what to do with the request.

Most of the time, the request is responded to by the server sending out the contents of the website you asked for, but there's a lot of different ways the information can be handled. You can have a web server that delivers web pages. You can have an email server that sends and receive email messages that you then can read on your **email client**. You can have servers that move files (**FTP servers**), and servers that manage people and their authorizations (**Identity servers**)[27].

In order for a website to be visited by a user, they first have to go to their web browser. They use the web browser to ask for a webpage using a **Uniform Resource Locator (URL)**, which is an address that tells the browser which server to contact. The browser passes the request to the server. The server responds by sending the contents of the web page to the computer. The browser then interprets the data and displays it. This is why some websites look different using different browsers.

What about open-source servers?

OSS Server software can be downloaded in a bundle to quickly get a server up and running. The bundle is often called **XAMP** (or something similar). The different letters refer to different software included in the bundle. The first letter is the indicator of what operating system the bundle is designed to work with. W stands for windows. L stands for Linux, and X means it will work with a variety of operating systems.

The M stands for **MariaDB**. The DB stands for Database, and specifically what is called a **relational database**. A relational database allows some structure to the information. The information can have a relation with other information in the database. MariaDB allows information on the server to be structured and queried.The A stands for **Apache**, which is a **Hypertext Transfer Protocol Server (HTTP)**. You may recognize the HTTP from the begging of websites. Think of HTTP as the language of the internet. It's the written

27. https://www.lifewire.com/servers-in-computer-networking-817380

language that computers and servers use to communicate back and forth. Apache runs the communication on the server end. A web client runs the communication on the computer's end.

The P stands for **Hypertext Preprocessor (PHP)**. PHP is a server-side programming language. The P can also stand for Python, or **Perl** which are also OSS programming languages. If there are two P's, then there will be two programming languages installed.

At the end of this chapter, during the learning exercise, you will get an opportunity to experiment with a XAMPP (Operating System vestal, Apache, MariaDB, PHP, Perl) server on a flash drive. Even though it will be on a flash drive, it will run just like a XAMPP server would run in the real world.

The M in XAMPP *use to stand for* MySQL, *but that was acquired by a company, which made the* OSS *community wary of using it. MariaDB is a fork or branch off of* MySQL. *More about forks for software in later chapters*[28]
.

Open Source Identity management: Shibboleth

There are thousands of OSS designed to run on a server in the XAMP environment. To highlight one relevant to libraries, we want to bring your attention to **Shibboleth**. Shibboleth is an identity

28. https://www.computerworld.com.au/article/457551/dead_database_walking_mysql_creator_why_future_belongs_mariadb/

management software known as a **federated identify solution** [29]. If you have ever had to use a user name and password, then you have unknowingly been using identity management software. It keeps track of people, their **credentials**, their rights and abilities in a particular system. Shibboleth, in particular, is designed to be used in lots of different places, and not just a single site. This is often called **single-sign-on**. If you have ever used your Facebook or Google account to log into an app or website, then you have used a single-sign-on identity management system. Shibboleth has been mostly adopted by the education and researcher communities.

Libraries have many opportunities to use an OSS like Shibboleth. A library can set it up so that users can log into many different databases with a single log in, or they can use it to manage their **institutional repositories** and who can add content or edit.

A lot of different library services require log in, and many new ones will as well. It's nice when someone suggests a new service if you have a Shibboleth instance that can just be hooked in. It makes the patron experience easier just having to log in once.

Reflection Exercise

When you go to a website like Wikipedia, you are actually working with all the software mentioned in this section. Shibboleth itself isn't used by Wikipedia, but the software has its own built in identity management system. Think about all the people accessing Wikipedia right now. The English site gets 18 billion views per

29. https://www.shibboleth.net/index/

month [30]. Imagine the server software having to keep track of all those people coming in all at once. The server has to hear thousands of request from user computers and respond appropriately to get everyone the pages they wanted. When the number of requests gets too much, the server crashes. If this is done on purpose, it's called a **Denial-of-Service-Attack (DoS)**. Sometimes, this is accidental, like when a celebrity dies and everyone rushes to a website to read the news.

Think back on the last website you visited. Was your experience of these server level software seamless? Did you have any idea that you were using a suite of specialized software? Now that you know, do you feel like it changes how you view websites?

Section 3: Top of the Stack

Section Summary

The top of the stack can be thought of as the front-facing part of all the software mentioned previously. It's the part that you see. In this section, we will discuss the topmost layer of development,

30. http://www.pewresearch.org/fact-tank/2016/01/14/wikipedia-at-15/

and how that relates to libraries. We will, specifically, talk about library-specific software because in many cases it is designed to be forward-facing to the patron.

Website Software

If you have a XAMP server running, you still have to design and present a webpage in order for it to be presented to the computers that request it. This is done through a **hypertext markup language (HTML)**. A markup language is a way of annotating text, images, and other things so that a computer can treat different parts differently.

Example:

This is just regular text.

<b>This is text annotated with HTML code that lets the computer know to make it bold.</b>

This is what you would see on your web browser.

This is text annotated with HTML code that lets the computer know to make it bold.

The mechanism by which HTML works is simple, but it can be combined into a very complex website.

However, it gets tedious and difficult to maintain when you have lots of HTML web pages. Say you have five hundred webpages, and you want to change the way your bold text looks. You will have to go through each one and change each instance where you used the bold tag.

That's where the concept of **cascading style sheets (CSS)** comes in. The idea is that you have a single document that says how each element should be displayed. So, if you want to change all of your bold text to bold red text, you can do it quickly by just changing the one document.

HTML and CSS are not OSS. They are actually standards, not software, so they can't have a license. However, if the software lives online, then it more than likely has an HTML or CSS layer involved.

Let's say you want to do something more complicated than just simple formatting. If you want a form that people can fill out and send you an email, you can use **JavaScript**, which is an OSS programming language specifically for use on HTML web pages.

Not everyone who works on web pages wants to learn how to be a HTML, CSS, or JavaScript master. Instead, they want to focus in on their content and making it available. This is why a whole category of software was created to make creating and editing web pages as easy as writing a word document on your computer. This is where the content management systems (CMS) come in.

One of the most popular OSS CMS (lots of letters here), is **Drupal** [31]. Drupal, while OSS itself, can run on both OSS server software and proprietary server software. Drupal adds features that allow a website to be more easily run, like user accounts that let multiple people edit the site.

Libraries need websites to tie their online resources and their library information together in one place, and many libraries choose Drupal to do that. There are even some **modules** designed specifically for libraries, such as Drupal OPAC, Article discovery layers, and digital asset management systems (DAMS). Islandora is a image repository software designed with specific digital library software and it uses Drupal as the front-facing user interface.

Library Enterprise Software

The biggest challenge for libraries and OSS is the idea of **enterprise-level software**. This is software that is designed to handle the entirety of an enterprise. The most well known and talked about kind of enterprise-level software is the **integrated library system (LIS)**. An LIS is a single monolithic piece of software

31. https://www.drupal.org/

that handles most of the functions of a library, like acquisitions, cataloging, and circulation. Sometimes an LIS will handle things like digital collections, course reserves, and interlibrary loan.

Most of the literature in libraries about OSS have to do with moving to an OSS LIS. The two most popular are **Koha** and **Evergreen**. These two have had the most written about them, and the most number of libraries trying to migrate to them from various systems. Koha is the more mature software, but it's limited in that it doesn't work well for multiple libraries. Evergreen was developed with library consortia in mind, so it handles various libraries' sites a lot better.

Moving to an OSS LIS can be difficult, and the choice to move would be one that is not lightly undertaken by an organization. The very nature of LIS means that it touches nearly every part of the library and nearly all patrons, so changing from one system to another affects everyone. In addition, staff will most likely have to learn new workflows, and counterintuitive design because these systems are, by and large, not designed by librarians. Handling the choice of whether or not an organization will move to OSS will be handled in Chapter 3. The argument for why librarians should be involved in the development of OSS, and how to get started, will be discussed in Chapter 6.

For your reference, table _ is a listing of well used OSS software in libraries.

System	**Purpose**
Koha	LIS
Evergreen	LIS
Backlight	Discovery Tool
Extensible Catalog/XC	Discovery Tool
VuFind	Discovery Tool
Samvera	Repository Software
Omeka and OmekaS	Digital Collection Software
DSpace	Repository Software
Islandora	Digital Collection Software
Fedora	Digital Asset Management Software
Greenstone	Digital Collection Software
DataVerse	Data Management Software
OpenILL	Interlibrary Loan Software
FillfILLment	Interlibrary Loan Software
OpenRequest	Interlibrary Loan Software
Prospero	Interlibrary Loan Software

In the learning activities, you will get the opportunity to play with Koha without having to install any software, or do anything more than go to a website. For every other system, you would have to work somewhere to get experience with an ILS, but with OSS some places will make example sites available for experimentation and learning.

Reflection Exercise

For every website you visit, someone has had to work to make it look the way it does. There was programming involved in how the site functions. How comfortable did you feel with the top of the stack technologies? Are you curious about doing more or are you glad the interactions between the different levels are seamless?

Chapter Summary

In this chapter, we went through a whole stack of software from the very bottom to the top. Technologies you were probably not familiar with, and technologies you probably use every day without realizing it.

Links to External Resources

Open Source Hardware Association (https://www.oshwa.org/)

Arduino Tutorials (http://www.ladyada.net/learn/arduino/)

Learn Java (https://www.learnjavaonline.org/) Has tutorials and an in browser code testing space so that you can learn and test Java without having to download or run anything on your local computer. The tutorials have basic programming concepts and some advanced concepts.

Learn Python (https://www.learnpython.org/) Same group that does the Learn Java, but without the in browser tester, so you will have to install Python to get to test on your computer.

Drupal for Libraries (https://groups.drupal.org/libraries/resources)

Self Assessment

1. Why is Python a good programming language to start with if you want to learn to program?
2. Explain how a server interacts with a computer to get a webpage.
3. Why would you use a Content Management System?

Exercises

One of the great things about open-source software is that they are all very available. You can install them and get hands-on experience with them immediately. Depending on your level of technical expertness, the learning curve might be smoother or steeper, but it's all available at your fingertips. In this section, the authors have chosen some small projects with minimal resources to do introductory tasks. Getting experience with these systems and software now can help you inform your IT decisions in the future. Based on your experiences with these tasks, is proprietary software worth it?

You can attempt any one of these or all of them, but as you work through the process, keep a project journal. Keep track of what you are trying to do, links that you used, notes on how successful or not you were, and track how much time it took you for each task. Whenever you are working on a complicated project, keeping this kind of journal will help in re-tracing your steps, or figuring out where things went wrong. The success criteria for these are that you attempt them. Trying and failing is a part of learning, so don't shy away from something just because it looks hard. There are different activity review points if you fail. However, to prove you tried, you should still keep your project journal of what you tried and how you found information, what worked and what didn't.

Get experience using Linux

The goal of this exercise is to install some form of Linux and use it to get to the internet

Install Porteus, a portable distribution of Linux that can be installed on a USB or CD (http://porteus.org/). This distribution is not setup, for example, to host sever level software, but to get on the internet and maybe write a document. It's an incredibly small distribution of Linux, and it is suppose to boot very quickly.

One of the benefits of this is that you can use a computer that runs Windows to set it up, and view it. The downside is that it's a bit of a difficult mess to try to install. Just read what ____ said about their attempts to get it running : http://dailylinuxuser.com/2016/09/a-not-for-everyday-linux-user-review-of.html

If you succeed in installing Porteus and making it to the internet, write about what you think about the interface, and the experience. What other software did you choose and why? What do you think of the other software packages that come on this distribution?

If you are not successful in installing Porteus, write about your experience in trying, what you tried, and how it failed. Write about how the process can be made more user-friendly. Would more documentation help? More detailed instructions for how to start? What resources did you look at to try to find answers to your solutions?

Get Started with Python

The goal in this learning activity is to learn what it takes to start programming in an open-source programming language. The language of choice for this activity is Python. The task is to use Python to print "Hello, World!". Printing a command is one of the simplest tasks in any programming language, and getting to "Hello, World!" on your screen can be a journey all its own. It requires you to install the programming language on your computer, understand how to get to the command line, and to understand the syntax enough to tell the computer what to do. Thankfully, there are plenty of instructions on how to get started in Python.

Go to the Beginner's Guide for Python and try installing it from those instructions: https://wiki.python.org/moin/BeginnersGuide/Download

https://www.python.org/about/gettingstarted/

To learn the syntax, you can go to https://www.learnpython.org/en/Hello,_World!

There, there are tutorials for how to print out the welcoming phrase in different versions of Python.

If you succeeded at getting the program to print "Hello, World!" write about your experiences getting to that point. Was it easy? Was it difficult? What resources did you use to help you to this point? Do you think the average library science student can get to this point?

If you failed to print the term, write about your experience. What failed? Did you give up? Did you run out of time? Does this experience make you have a new appreciation for developers? What could have made this easier? Go to Learn Java (https://www.learnjavaonline.org/) and try to get it to Print “Hello, World!”. It’s as simple as hitting the green “run” button. You didn’t have to install any software and technically completed the task. Try editing the code to print something else. Do you think more people would code if it were made easier?

Get experience with LibreOffice

Download and install LibreOffice. Try to use the software instead of your normal software for a few days. How is it different from what you normally use? Are you impressed with it or disappointed? Are you likely to continue using it or are you more firmly set on using your software of choice?

There are no wrong answers. Getting experience using open source software can be an eye opener into what is and isn’t possible with collaborative programming.

Get Experience with Server Level Software

Install Mediawiki using Wiki on a Stick : https://www.mediawiki.org/wiki/Manual:Wiki_on_a_stick

This whole process is a simplified version of installing a real server with a real Mediawiki installation. The only difference is that you are installing it on a flash drive or your computer hard drive instead of on a server. Because it's localized, you won't be able to have other people use it, but you can get the experience of setting it up and using it. You will actually have to turn on the surver, have it boot up, before you can edit your Mediawiki instance. Write about the different open source software that is involved in setting up just a simple wiki, and how difficult or easy the process was.

Or, install Drupal on your local machine https://www.drupal.org/docs/develop/local-server-setup and create a simple webpage that says "Hello, world".

If you are successful with either of these, compare your copy to a version of the software running in the real world. Compare your Mediawiki instance to the real Wikipedia, or compare your Drupal to a library website using Drupal (choose one from this list: https://groups.drupal.org/libraries/resources). How much time do you think it takes to get from a fresh install to a website full of content? Do you think one person can do it alone, or should there be multiple content creators?

If you were not successful, what do you think was your limiting factor? Did you not have enough time? Did you not find the resources you need when you ran into a problem? Delve deep into how you think the community could provide better instructions.

Get experience with library-specific software

Go to the Koha Community Demo page (https://koha-community.org/demo/), and choose a Demo site to use for both the OPAC and Staff Interface. In the OPAC do a search for "War and Peace", then do the same search in your normal OPAC of choice. How do they differ? Are you impressed or disappointed? Why?

Go to the staff side using the demo site's log in. Look through all the modules and play around with them. See if you can find patron records, and see if you can check out a book to them. If you have taken a cataloging course recently, try to catalog a book from your private collection into the system. For some people, this may be their first time getting on the backend of a library system. For those people, write about your experience and if it was harder or easier than you imagined. For those who have experience with other systems, how does this system compare to other systems you have used?

Compare DSpace and Greenstone

Go to the Greenstone examples page (http://www.greenstone.org/examples) and look through a few of the highlighted Greenstone collections. Then, go to the DuraSpace Use-Cases for DSpace site (http://www.duraspace.org/dspacedirect/about/use-cases/). Based on a review of three different sites for each system, which system looks more modern? Now, go to the Samvara Partners page (http://samvera.org/samvera-partners/) and find at least one example collection to compare to the others.

How has the user experience for digital libraries changed from the first generation of digital asset management software to the newest?

Get a feel for an open-source discovery platform

Go to the example pages for Blacklight (http://projectblacklight.org/#examples) and choose one of the project showcases. Compare that to the out-of-the-box demo at https://demo.projectblacklight.org/

How much customization did different organizations put into their OSS? How many programming hours do you think it took to take Blacklight from the demo to one of the live projects? If you were a library administrator seeing some other organizations' Blacklight interface, do you think you would be happy with the out of the box version at your own institution?

Can you afford open-source hardware?

Imagine you are starting a library maker space and want to purchase some Arduinos. Do an evaluation of what different versions of Arduino are out there, how much they cost, and any accompanying hardware or software that you might need to have patrons use them. What would be good first projects to do as part of an Arduino workshop? Act like you are writing this up as a proposal to your director or dean to try to fund the project.

What argument can you make for why a library should be doing this type of workshop? If you were in charge of a library, do you think you would be convinced?

Self Assessment Answers

1. Why is Python a good programming language to start with if you want to learn to program?

It's clear readable syntax, quick progression, versatility, available open resources, and supportive community make it a great first language.

2. Explain how a server interacts with a computer to get a webpage.

   In order for a website to be visited by a user, they first have to go to their web browser. They use the web browser to ask for a webpage using a Uniform Resource Locator (URL), which is an address that tells the browser which server to contact. The browser passes the request to the server. The server responds by sending the contents of the web page to the computer.

3. Why would you use a Content Management System?

   A Content Management System simplifies the creation of websites so that users don't have to understand the various technologies involved in order to create web sites. A Content Management System can make it easier to maintain a website by making it easier to edit content, and manage multiple people editing.

3. Contributing to Open-Source Projects

Learning Objectives

- Learn how open software development projects work
- How to contribute to open-source software if you're not a developer
- How to start on the path to becoming a developer that works on open source projects

Introduction

Open-source software is about the whole community and not just developers. In fact, open-source software is made better by engaging a variety of people in the development process. This chapter will help you understand how you can contribute to an open-source project at any level of experience and expertise. For those of you who want to take the plunge into open-source coding and development, the end of the chapter has a guide on how to get started.

Focus Questions

1. What are ways that anyone can contribute to an open-source project, even without knowing how to program?
2. What is a way you can engage with the open-source community today?
3. Why are open-source projects difficult to maintain?

Engaging with the community

Section Summary

Getting involved with open-source software can be as easy as using the software and talking about it. This section goes over some soft ways to participate in the community for people at all skill levels, without having to know a single bit of code. This section borrows from an article called How non-programmers can contribute to open source projects by Duncan McKean from October 24, 2013.

Using Software

The first level of involvement with the open-source community is the simple act of using the software. Find a program that is open-source that you like, that you can use, and start using it regularly so you get to know it. This could be as simple as a word processor, or something more involved like a video or audio editing program.

You should approach using open-source software differently than using proprietary software. With proprietary software, you can install it and use it without really knowing how it's made or how it is developed. You might encounter a problem and then just give up and find another way to do what you want. However, to get the most out of open-source software, and to give back to the community that made it, it's best to be an active user. This means not just using the software, but understanding a little about how it's made, how to report problems, and maybe understanding the community that makes it a little bit more.

The next step is to start becoming more familiar with the community that maintains the software. Often, these projects have websites that describe the project and how it's maintained. We will use Wikipedia as an example since most people are familiar with it.

Wikipedia is run on a piece of software called MediaWiki. You can find it in the link at the very bottom of any Wikipedia article. The MediaWiki project page describes the project, has instructions for how to set up and install MediaWiki, how to edit and use MediaWiki, and then a guide on how to develop and extend the code. They even have helpful guides on how to start coding for a Mediawiki project if you are new to development. The project page also has information for if you are already a developer. The site also has news that describes updates to the software and what was included in that update. A good first start for getting involved in open-source software community is to start reading these updates and understand how software develops over time.

Join a group

Join a membership, or just a users group for a particular piece of software and follow the discussion. Answer questions if you can. Post questions if you run into problems. Steering committees, standards group, advisory groups, working groups. Sometimes, just being at the table to discuss things is a huge help to the community.

Bug Test

As you use a piece of software, you will inevitably find problems with it. That is actually a great thing because you can then provide that information of the problem you encountered back to the community so that someone can fix it.

This requires you to find the places where that software's community discuss bugs and problems. For the Mediawiki page, they have a bug tracker called Phabricator. On the page for the bug tracker, the community has guidelines for how to report software bugs, how to report security issues, and even how to request that a new feature be created. Have you ever sat down with a piece of software and wished it would do something specific for you? With open-source software, it is possible to request these of the greater community. It might be that other people also would like that enhancement, so you might get what you want.

Write Documentation

With proprietary software, you are almost always guaranteed to have documentation already written when you install the software. With open-source software, often documentation is the last step in the process. Developers are busy writing code, and many don't have a passion for technical writing. A simple and effective way to contribute to these projects is just by going over the documentation and making it better, making it more clear, adding parts that are missing or writing out tutorials.

Other ways to contribute.

If you have experience or expertise in more than one language, then you might be able to help with translating either commands in the software to a new language or translating documentation.

If you are a librarian who does user experience (UX) work, you could do some UX testing on the software and provide that as feedback to the developers.

If you are a designer, you could actually help by designing the interface to be more appealing or modern.

If all else fails, and you have the funds, you can often support a open-source project by donating. Mediawiki, for example, is run by the Wikimedia Foundation which has goals to support education worldwide and fighting legal battles to make free knowledge possible around the world.

Development for Open-Source

Section Summary

In this section, we will talk about more direct ways of contributing to open-source software, like learning to code. This section borrows from A developer's guide to getting into open source by Radek Pazdera from Feb 19, 2015

Learning to Code

You don’t have to have a computer science degree to code. The degree does help people understand the more complex aspects of software development, and it gives them forced practice. However, much like art, you don’t need a degree to be good at it and contribute to a worthy cause.

In an ideal world, you would find a project you want to contribute to and start down the process of learning the language it’s written in. However, not all languages are easy to learn. The ideal first language would be one that is relatively easy to learn, with lost of open resources so you don’t have to spend money to learn, and also teaches you the basics of programming. The language we recommend is Python. Python is a very versatile language that is designed to be structured in a way that makes the syntax easy to learn, and easy to read. Because it’s an established open-source programming language, there are a lot of high-quality resources out there.

If you want to get started in Python without having to download anything onto your computer, you can use the w3Schools’ Python Tutorial. This will get you a good introduction to the language, some experience coding, and all without having to download a single thing onto your computer. However, to get the most out of it, install Python on your computer and use the things that you learn in the tutorials to make a simple application. Try to use everything you learn. The application doesn’t have to make sense, it just has to help you remember what you’ve learned. Instead of paying for the certificate, try to start making your own portfolio of your learning.

Open book project: How to Think like a Computer Scientist: Learning with Python 3

http://openbookproject.net/thinkcs/python/english3e/

Programming is all about learning the logic of how things work in programming. Once you’ve got a solid foundation with Python, you can then look into what projects you are interested in contributing to, and what language they are coded in, and you can find tutorials on those languages. You'll find a lot of the concepts are the same, but the syntax that you use to express them are different for each language.

Choosing a Project

One of the most important parts is choosing a project that you care about. Preferably, it should be a piece of software that you use often and are familiar with. Being invested in the product will help your motivation stay high.

Learning the Code Base of a particular project

There are lots of good guides on how to get familiar with a codebase in general including the article: How to get familiar with a new codebase by Perikis Gkolias, published March 28, 2018. In this article Gkolias talks about how this is a skill just as every other part of development is a skill, which means it can be practiced and you will get better at it over time. The first step Gkolias recommends is to read all the documentation available. Look for documentation on project pages, wiki pages, or in the code repository. Next is to read any notes commented out in the code itself. If it is a large piece of software, then try to focus on one segment of it and figure out how that works, then move to another. Break up the code into digestible bits that you can fully understand.

The next step is to understand the code's dependencies by understanding how it is built. To build a piece of software is to create the environment it needs to run and add in all the software or mechanisms it's dependent on. Gkolias suggests that once you have the software successfully running, start using it as an end-user, and then look at whatever logs are produced. This will give you an idea of the order of things, or what sequences functions happen in.

Gkolias's next suggestion is that you start understanding the developer's logic of the system, and start mapping out or drawing out the logic diagrams. These are visuals that help you understand how information flows in the system.

After you have an idea of how the system flows, then you can start trying to solve a bug or a problem. Something small and manageable.

Start Small, form partnerships, develop a reputation

Once you understand the codebase and are familiar with how things work, the next step is to get into the loop with the other developers on the project. Developers communicate in different ways in different projects, so just try to find where and how people are communicating about that one. Join any mailing lists available. You don't have to read every email that comes your way, but you'll start to see how things work with the community. Don't hesitate to make direct contact with the developers or maintainers in asking how to get involved if the way is not immediately clear.

Make one small contribution, then make another

Read any code submission guidelines, and follow them. Follow any naming conventions that the community has set out, and in general make it as easy as possible for the maintainers to use your code. Don't get discouraged if someone reviews your code and rejects it. This is a learning process. Use it as a learning opportunity and take some extra time to try to understand why it was rejected, and get a sense for what the maintainers want.

Chapter Summary

Links to External Resources

A review of how most of open-source development is done by vendor supported developers: https://www.infoworld.com/article/3268001/open-source-isnt-the-community-you-think-it-is.html

Exercises

Document Writing Sprint

One of the problems with open-source software is a lack of documentation. One of the ways that communities try to address this is by having documentation writing sprints where a group of people gets together either physically or virtually, and contribute to filling out the documentation for the software. Sometimes this documentation is housed in a wiki, sometimes it's organized somewhere else. You can use the DuraSpace DSpace Wiki to see an example of how documentation is organized.

This is where you can add appendices or other back matter.