code{4}lib JOURNAL

ISSN 1940-5758

Issue 5, 2008-12-15

COLUMN: We Love Open Source Software. No, You Can't Have Our Code

Librarians are among the strongest proponents of open source software. Paradoxically, libraries are also among the least likely to actively contribute their code to open source projects. This article identifies and discusses six main reasons this dichotomy exists and offers ways to get around them.

by Dale Askey

Introduction

Not long ago, a public university library created Citation Builder, where users enter the pieces of a citation, click a button, and have them magically formatted as APA, MLA, etc. Slick, but not really revolutionary since others have done this before. Somewhat later, another university in the same state got their code, enhanced it, and rolled out their own edition, explicitly acknowledging the other university's work on the original product.

Still later, my library hired a librarian who recently earned her MLS at the university that had created the second version of the service. The new librarian offered to contact some people she knew at her degree school and see if we could get their code to use as a basis for our own tool, provided we promised to thank them and share any enhancements we developed. Sounds a bit like ad hoc open-sourcedness in the making.

Except that it wasn't. Said university responded by asking us to sign a software licensing agreement, which we refused to do because we don't like to sign licenses for such trivial things, and we certainly didn't want to be bound by any limitations on sharing the code further, since we intended to build significantly on their core. In the end we opted to develop our own citation generator. This is inefficiency piled on top of inefficiency. Citations and style manuals are constants; there is neither reason for nor benefit to having everyone create their own citation tool for their users. This kind of library-to-library software sharing saga is sadly not all that uncommon. (See Correction.)

Before offering any further observations on the ways libraries share and fail to share open source software, I should clarify my perspective on the topic. First and foremost, I am not a programmer. What I am is one of those librarians who is otherwise technically semi-competent, i.e., I can install software on Solaris/Linux, know what a shell is, sometimes find myself hitting :wq in Word, and can do something I'd refer to as sys-admin lite. I've also spent many productive and not-so-productive hours wrangling both open source and commercial library software, giving me a head full of lessons and ideas. This is not unique—there are many librarians who exist in this netherworld between IT work and "librarian" work.

The general topic of open source software is simply too broad to cover here. As Dan Chudnov points out in his chapter on open source software in the book *Information Tomorrow*, open source software has, at least on one level, successfully integrated itself into libraries, particularly when it comes to system infrastructure and programming languages (Chudnov 22). Additionally, libraries have strongly embraced certain consumer open source applications, such as Firefox and Thunderbird, as well as object repositories such as DSpace and Fedora and content management systems such as Drupal.

That's all fine, but where we tend to fall flat is in the area of creating, maintaining, and sharing library-specific applications. There are certainly myriad exceptions to this statement, but I would suggest that however large and noteworthy, they remain the exceptions, and not the rule. We also miss the boat on sharing what I'll call open source lite: tiny programs written in various scripting languages that drive all the doodads and widgets on our Websites, or extend (or, in some cases, repair) the functionality of our commercial systems. These are typically less than a couple of hundred lines and are often deemed lightweight and insignificant to those who write them, but are godsends for the many librarians who lack access to programmers. Collectively, they represent thousands of hours of valuable work time. Even those libraries rich in programming resources can benefit from not having to create everything on their own.

After pondering this issue for some time, I identified the following issues as the driving forces that undermine the sharing of open

source software in libraries:

- perfectionism – unless the code is perfect, we don't want anyone to see it
- dependency – if we share this with you, you will never leave us alone
- quirkiness – we'd gladly share, but we can't since we're so weird
- redundancy – we think your project is neat, but we can do better
- competitiveness – we want to be the acknowledged leader
- misunderstanding – a fundamental inability to understand how an open source community works

Many of these issues operate in combination, but any one of them is sufficient to thwart the development and adoption of open source software in libraries.

Perfectionism

Perfectionism is perhaps the most obvious issue that arises with open source software, and plagues mainly smaller applications that were developed by one or two people at a given institution. We have all encountered this type of programmer, where untidy code, lack of code commenting, and–sin of sins–inefficient or redundant code makes them unwilling to show it to anyone else who might be able to point out the flaws and correct them.

It may seem trivial to peg some of our woes with open source software on the programmers' desire to hide their imperfections, but the issue repeatedly arises. One hears it mentioned in conversations with programmers and reads it in the messages one receives in reply to a request to see this or that snippet of code that they have written.

Even at library conferences where many successful programmers congregate, I've heard people at the podium talk down their code and make self-deprecating jokes about it, even though it's perfectly functional and useful. There is always that fear that someone else in the room knows it better and is silently smirking at the fact that the person at the podium didn't know this or that trick. Expand that audience to the wider library world, and it's a wonder that any less-than-perfect code ever gets shared.

Dependency

Somewhat related to perfectionism is dependency, which has the programmer and his or her time at its core. Nothing is more certain in the world than this: if you share software with someone, you will be asked to support it, even if you make it perfectly clear that you have no ability and no intention to do so.

This can be traced to how open source software moves through libraries. It begins with personal contact: Librarian X asks Programmer Y for some code. Flattered and temporarily overcoming their innate perfectionist side, Programmer Y happily complies. When something goes awry — inevitable due to the myriad system variables in play — Librarian X writes a "sorry to bother you, but" message. Being a nice person in a service-oriented organization, Programmer Y assists them, but may end up somewhat cranky about it since it detracts from their own work and yet does not add to their list of accomplishments.

There are at least two ways out of this conundrum. One is to accept one's fate and provide support more or less explicitly. Virginia Tech's LibX toolbar falls into this category. As an adopter, I'm certainly grateful to have a functional toolbar to offer users and an edition builder in place to create and maintain it, but often wonder about the sustainability on Virginia Tech's end.

The other path, the one that I wish more libraries would follow, would be to post the code in some public repository, along with at least rudimentary documentation (perhaps the work of one or two hours). That way, the code can be picked up anonymously, breaking that person-to-person chain. A public repository makes it clearer that the code is being offered as a service, but support may not be forthcoming. Needless to say, it should be posted with an unambiguous open source license.

This sounds easy enough to do, but it's not a reach to say that it rarely happens. Libraries typically do not make it a priority to share their successful tools with other libraries. New projects become the focus, staff changes, priorities shift; this renders this last, critical step expendable all too often.

Quirkiness

Perhaps most maddening among the various reasons for not sharing software is quirkiness, the sense that one organization's needs are so locally-tailored that would make no sense to release the software to the broader library community. One would think that for institutions with a high awareness of standards this would not be a major issue – but it is.

At a Coalition for Networked Information meeting in 2007, the University of Maryland presented the digital library that they had

built on a Fedora platform. The features and tools they had created were impressive, and I naively assumed that the question I posed was an obvious one, namely, when did they intend to release the code for others. What they offered was a reason why they could not: they had used an idiosyncratic local metadata scheme. If they had to do it over, one of the presenters offered, they might reconsider that choice.

This was an unwise decision on their part. They now have applications that they must support on their own, entirely at their own cost. Should the developers leave, they may struggle to find others to maintain it. Even if they do, will they find someone content to babysit someone else's application, or who wants to reshape it according to the context and frameworks they know best? They also lose any potential benefits that could accrue from having others extend or refine the functionality.

I have some sympathy for choosing this path. Programming is neither a trade nor a rote process, it's a creative art. There are myriad ways to solve problems, and a bevy of languages and toolkits that can be learned and applied. Babysitting someone else's code, or worse, being asked to clean up someone else's code written in a language one doesn't know well or doesn't like, is thankless and uninspiring work. It is much more interesting and enriching to create applications and build something that is one's own.

Redundancy

The flip side of quirkiness is redundancy, which is when there is perfectly acceptable software available and yet is rejected because it's not quite what one would have done had they created the software. In simple terms: reinventing the wheel. This could also be called the "big challenge" dilemma. Librarians who are not programmers or system administrators want services, and want them delivered on time via the path of least resistance. Programmers, on the other hand, are typically systems thinkers, with strong opinions about how best to build and maintain a system.

Perhaps a bit of arrogance is also in play. It's tempting to look at software, identify its flaws, and declare that one could do better. However, the reality is that libraries and universities have limited resources, and often have to make compromises to be able to sustain all of their services. It is more than likely that the software that results from this line of thinking will evince different, but equally obvious, flaws.

How many of us work at universities where some IT entity or authority has declared that the software available for a given task, whether it be commercial or open source, is unsuited to the unique situation at hand. What can ensue is a multi-year exercise in wheel reinvention, which often does not end happily.

My university has fallen into this trap. It refuses to sign on with iTunes U as a podcasting platform, choosing instead to write its own platform to compete with Apple. A press release issued in 2006 was subsequently picked up by *The Chronicle of Higher Education*, which declared that Kansas State was about to "become the educational-podcasting capital of the world" (Read). Not long after that, someone from the office tasked with developing this wunder-platform remarked sarcastically in a meeting that "we don't need no stinkin' iTunes." This was over two years ago, Kansas State isn't even the podcasting capital of Kansas, and faculty still clamor for iTunes U.

Moving back to libraries and to open source software, I would suggest that one concrete manifestation of the "big challenge" dilemma is the emergence of the Open Library Environment project. Its goal is to define and develop an open source integrated library system geared toward academic libraries. There are, however, academic libraries who have either implemented or are about to implement an open source ILS, as well as a healthy number of active developers for this particular ILS. While it would be mean-spirited to wish the OLE ill, it seems unwise to fragment what fragile momentum there is toward the open source ILS in academic libraries by creating a new fork.

Competitiveness

One could also view the OLE as an example of a bit of competitiveness. Competition can serve a purpose, driving innovation and development. It can also sap resources, and in the context of libraries, where we have no access to venture capital, clearly collaboration is to be preferred to competition. Most libraries do not have a budget line item for research and development, and so we scratch together bits of money to drive our development projects. While grant-funding, such as the Mellon money driving the OLE, is a wonderful thing, it comes with its own problem, namely, how to sustain the project beyond the scope of the grant. Having a limited set of partners is acceptable while the money flows, but you need a mass community to keep things going.

At least within the United States, much of the competitive spirit between universities arises, sadly, from our sense of athletic competition. Practically every institution is interested in building its brand and creating its community, all in the name of generating revenue for the institution. As such, it is more the exception than the rule to see substantial collaboration between universities as part of their normal way of doing business.

This certainly extends to libraries, where we all dutifully build our own institutional repository, our own digital libraries, and develop

our own Web services for our users, typically with little regard for what's going on down the road. How many libraries have invested how much labor in getting an institutional repository off the ground? What results has this intense investment brought most institutions? These are questions we do not ask for the most part. We need to move past the notion that partnering and using the tools at hand is tantamount to capitulation.

Misunderstanding

Perhaps the most vexing issue that plagues open source software in libraries is a simple misunderstanding on the part of many librarians and library administrators about the nature of open source. Many librarians have acculturated to an environment where our software needs are met by using a product from an established vendor that has a large booth at our major meetings. Any new library technology that arises predictably leads to the development and marketing of multiple commercial 'solutions.'

On the one hand, this is understandable. Licensing a product from a firm removes many of the variables, provides stability, and does not require hiring too many people into libraries whose skill sets the typical librarian does not understand and cannot assess. On the other hand, we have a culture of complaint where we expect vendor software to react immediately to our every whim and fancy. This is, of course, unrealistic, and leads to myriad enduring versions of the "our OPAC sucks" meme.

Open source clearly represents the remedy to this issue. If you need functionality, find software that comes close, hire some programmers, and get to work. Find partners, share code, collaborate. In the end, you significantly shorten the feedback loop between users (whether they be staff or the public) and developers, and can exercise far more control over the software than with a vendor.

Sounds simple, but it often fails because of the innate culture of many libraries. Having become mired in the vendor-driven rut, we cannot find our way out. In that world, we are simply consumers of software. We pay our money and get a product. We provide feedback, ask for enhancements, but that is more or less the extent of our involvement in the development of the software.

With open source, one can choose simply to download the code, install it, and start using the software. That's not likely to get one very far. There are versioning issues, customizations, security concerns, backup plans, bugs, and so on. What often happens with open source software in libraries is that non-technical librarians and staff begin to view their colleagues down the hall in the IT shop as the vendor, and simply continue reporting issues and asking for enhancements. What they fail to understand is that with open source, they have the ability and, I would argue, the responsibility to seek out their own solutions and participate in their implementation. They need not know all the technical details, but they should be able to articulate what they need, seek solutions from the community, and suggest how these would fit within their institution's suite of services. They should assist with usability and documentation issues, which do not require programming skills.

Essentially, it's at the point of shifting from being a consumer of software to being an active participant in the open source community that the misunderstanding hits its zenith. Most librarians do not understand the open source software model and fail to recognize that by using open source software, they are tacitly declaring their membership in the community. An example may help illustrate this point.

As reported in Technical Services Quarterly, Karen Calhoun presented ITSO CUL, the Integrated Tool for Selection and Ordering for Cornell University Libraries, at the ALA conference in 2004. Not surprisingly, she was "asked if Cornell would share the program with other libraries." She responded by asking if the audience would prefer an open source or a shareware approach. The audience favored shareware, "not only for the benefits to those in the audience who wished to use the program, but to Cornell by reducing their administrative costs and responsibility for the program" (Coulter 82). This shows a clear misunderstanding of both open source and shareware. The shareware paradigm squarely puts the onus on the developer to support and develop the software, since the idea behind shareware is to download and use the software as is, either as a trial version or with a request to pay a small amount to the developer. It would be the open source path that would reduce Cornell's burden, since some adopters would become partners in the ongoing development.

Not surprisingly, neither of those options ended up being the path taken. Instead, ITSO CUL morphed into WorldCat Selection, a fee-based service of OCLC. While one could correctly argue that this makes it more immediately accessible to more libraries since we're quite accomplished at writing checks to OCLC, in the long term it just adds another maintenance bill to the library's budget, and consumes resources that could be used for other things, such as a programmer, who could work on many projects. Perhaps more importantly, it also puts the ongoing development in the hands of a single vendor. In libraries, we certainly have extensive experience with the outcome of that scenario.

What Can be Done?

While it's good sport describing what is wrong with our use of open source software, one should also suggest ways out of the jam.

A simple way would be for us to agree on how best to share software between libraries. In this case, I would distinguish again between “macro” open source and open source lite. The former concerns large and robust enterprise software, such as Evergreen, Koha, OJS, DSpace, etc. Those clearly have their own user communities and distribution channels.

With open source lite, it’s much more Wild West. Some of it can be found on Google Code, some in Sourceforge, but most of it lives in little pockets at hundreds of libraries, and we pass it around as email attachments. That’s not terribly effective.

One promising development in this regard, which also happens to be an example of open source overlapping with commercial code, is the recently announced EL Commons, which appears to be intended to be a clearing house for user-created code that enhances Ex Libris products such as SFX. Having such product-centered spaces makes a lot of sense on many levels. One concern with such vendor-supported spaces is that they are rarely open to non-customers, out of fear that the tools might reveal proprietary information. At the very least, libraries who deposit code in such places should offer it elsewhere to the broader community, whether the vendor approves or not.

For the rest of it, should we just avoid reinventing the wheel and use Google Code? One could do worse, but that doesn’t make it easy for the non-technical to access. Google Code is certainly findable, but sites such as Sourceforge and Google Code use a language and presentation style that tends to repel the non-technically inclined librarian, i.e., most librarians.

One solution would be a library specific code repository. That seems an anathema even to me, since it’s potentially redundant given the options already available. If we want our not-so-technically bold colleagues to participate, however, perhaps it’s a necessary evil to lower the threshold for them.

Even if we lack such central points, libraries need to develop a far more open-minded attitude toward sharing their open source lite. Going back to my opening vignette, I would ask what the purpose is of asking a library to sign a licensing agreement for a couple hundred lines of PHP, other than to demonstrate how beholden our thinking is to our generally risk-averse library culture and our vendor-oriented business model. If someone asks for such code, shouldn’t the response be, sure, here you go, give back any improvements. Let’s be nice about these things, slap some sort of standard open source license on our software, and pass it on without the pointless negotiations or stern warnings. This step is nearly always skipped at a project’s end.

More fundamentally, libraries that wish to use open source software need to understand the staffing commitment they are making by going that route. Open source software requires programmers, interface designers, and system administrators. These people are often not librarians, although one frequently sees job ads asking for librarians who have such skills, as if library schools were actually cranking out such people in any quantity (they should be, and this failure will negatively impact libraries for decades). As such, we struggle as organizations to define the needs, properly prioritize them, and make progress on finding the right people with the right skills. We don’t commit to paying the salaries necessary to attract and retain enough experienced and talented people.

A related issue is the question of recognizing and rewarding staff contributions to the open source community. Libraries typically have no reward systems for putting something out there that helps others.

These are largely questions of leadership in libraries and the necessity of moving past the traditional roles and obligations of libraries and embracing radically different organizations than we currently have. Leaders must have a new vision of using large software packages, where the question isn’t simply open source vs. commercial, but, if one chooses open source, how to provide the resources necessary to make it successful. As Dan Chudnov pointed out in the conclusion to his chapter on open source software, in 2007, it is “impossible to run a library without software, but you can have a library without a building” (29). This insight should force us to significantly revise how we set priorities as organizations, but that shift has been slow to develop.

Why is all of this so important? The central concern I have is for the financial future of libraries. Surely all libraries face chronic budgetary concerns. Every time serials go up or our funding agencies issue cuts, our ability to do the things we lay out in our strategic plans contracts. On the IT side of the organization, we often do not get approval to recruit for the positions we need, or even if we do we can’t offer sufficient salary to fill them, at least not for very long. And trading the acquisitions budget for software purchases and maintenance contracts just seems like recreating the problem.

It’s often said that we need to become more efficient organizations. The IT side of libraries is one of the more underexplored areas for efficiency, given its relative youth on the budget sheet and the fact that some, shall we say, legacy administrators have difficulty understanding the IT puzzle, let alone how to create efficiencies. Open source is clearly part of the answer to this dilemma. While in the short term it requires new investment in staff—although one could point out that it’s really a zero sum process of converting vacant legacy positions into IT or programmer positions—offering better services free from the morass of maintenance and purchase contracts that accompany traditional vended software puts libraries in a much better position moving forward.

Notes

Chudnov, Dan. “The Future of FLOSS in Libraries.” Information Tomorrow: Reflections on Technology and the Future of Public and Academic Libraries. Medford, N.J: Information Today, Inc, 2007. 19-30.
(COinS)

Coulter, Cynthia M. “Acquisitions Technology Trends: Changing the Way We Do Business. A Report on the ALCTS Acquisitions Section Technology Committee Program. American Library Association Annual Conference, Orlando, June 2004.” Technical Services Quarterly 22.4 (2005): 77-82.
(COinS)

Read, Brock. “The Wired Campus: A Harvest of Podcasts.” Chronicle.com 27 Sep 2006. 6 Oct 2008 http://chronicle.com/wiredcampus/article/?id=1599.

Correction

7 January 2009: Actually, the colleague mentioned here pointed out to me that after working with both the second-named institution and the originators, we eventually did sign a license, albeit with the institution that originally created the software. We have been working with and adapting the original code, which is a good thing, but the fact that a license intervened means that this is still not open source when it could/should have been. Many apologies to my colleague and for any confusion my telling of the story caused.

Acknowledgement

This article is based on a talk given at Access 2008 in Hamilton, Ontario, on October 4, 2008.

About the Author

After stints at Washington University, the University of Utah, and Yale University, Dale Askey currently fills the role of Web Development Librarian at Kansas State University. On the information technology side of the library, he specializes in interface design and project management. He also works actively on scholarly communication issues, serving as publisher for the open-access journal imprint New Prairie Press.