The Emperor's New Repository

I don't know the first thing about building digital repositories. Maybe that's a strange thing to say, given that I work in a repository development group now, and worked on the original DSpace project years ago, and worked on a few repository research projects in between. If that qualifies me to say anything about repositories, though, it's just that I don't know much about what I'm doing, and I don't think many other people do, either.

I'd better qualify that some. It's not that there aren't smart people working on repositories -- there are plenty. It's not that few repository projects have good, important objectives -- many do. And it's not that we haven't learned anything in the past 10-15 years since "digital repositories" grew from a buzzword into a strategic program in many libraries -- we've learned a lot. But I don't know what this smart group of people with solid goals and lessons learned add up to yet.

Does that still sound strange? If it sounds strange to you, try this thought experiment. Say you get a new job as a director of a small library. In a new town. Without a library. So it's your job to build a library where one doesn't exist. What do you do first?

I've never run or built a library before, but I'd guess that you'd need to start with siting property and working with a city to plan infrastructure. Then come architectural design and construction bids, and when you're far enough along to plan what goes in the building itself you split out budget lines for staffing different departments, and furniture and shelving of several different kinds, and a floor plan, and meeting rooms and utility functions. Maybe you only can hire a few people, but you know you need to cover collection development, public and technical services, computing, accounting and payroll, and maintenance, whether with three people on staff or 30.

See? I don't know the first thing about running a library, but I know you'd better plan for at least all these things. You're probably thinking of

other things I didn't mention, whether you've ever run a library or not.

Now suppose you were hired to be a digital repository project director. For a new repository, which doesn't exist yet. What do you do first?

Eh?

I don't know, either. And that's what I mean!

Collect what you know

Given how long I've been around people and projects aiming to "build repositories", and how little confidence I have that we know what it really means to build a repository, I'd guess there are plenty of you who don't know, either. On one hand, sometimes it's okay not to know -- if you have a good goal in mind, like collecting faculty research, or making rare local materials available, the details of how you achieve your goal are less important than regularly measuring against the yardstick itself. I'd bet, though, that there are plenty of projects that don't even have this much clarity driving them. Absent such clarity, there are a lot of mistakes you can make along the way to achieving clarity in determining why you want a repository.

The first mistake to avoid is fetishizing software products or projects. Over the years I've had a lot of conversations with friends and colleagues who've asked "do you think Greenstone/Fedora/DSpace/EPrints/ContentDM/etc. is what we should use?" My answer these days is almost always the same: "it depends on what you're doing, but you can always start with one or another and decide after you have some experience, and they're all a good place to start, so just pick one and get started." There isn't any single answer and there isn't any clear winner. In an era when we're still trying to figure out what it means to build a repository, it's great that there are so many options, and it's wonderful that there are so many free software options. If you think you need specialized software to build your repository, then the key thing is to get started with a tool that looks like a roughly good fit. You're going to learn so much along

the way that the details of whether that tool's the best long term fit or not are going to become obvious to you as you build up experience loading your content and making it available.

The flip side of avoiding agonizing over which tool to pick is that you shouldn't hesitate, once a project takes some turns you don't like, to acknowledge that maybe you've made a bad choice with a particular toolkit, or that maybe you just approached the project the wrong way, with the wrong materials at first, or with the wrong staff, or even just at the wrong time. There's a software development axiom from Frederick Brooks, author of _The Mythical Man Month_, that applies well here: "Plan to throw one away; you will, anyhow." To plan for mistakes means to be ready to learn from them when they happen, and to minimize the cost in energy and expense when things go wrong. Start with a small collection, minimal staff, and a short timetable, and see what you can learn by building something quickly. All the feature comparison spreadsheets and RFPs in the world won't help you make a good decision if you haven't already started up the learning curve for yourself.

There's a broader point to remember here, too, that software isn't usually part of our collection development plans. We librarians know approval plans, cataloging standards, and search strategies, but selecting and implementing software isn't our strength. There, I said it: we're not good at this. But that's okay! Like a reference librarian assigned to select materials in a new collection area, we can learn as we go. The key steps are to get started, and to expect to make mistakes, but to be ready to learn from your mistakes.

I'll let you in on another secret of repository building. Adding new software isn't always the best approach to building a repository. Sometimes it's not only a mistake to introduce new layers of software between content and users or between content and staff, it's just a bad idea, period. You're probably comfortable, by now, with putting a web page online, and setting up a directory on a web server with some new files, or if you're not comfortable performing these tasks, you probably have colleagues or staff who are capable and experienced with basic web publishing. If you have a

small collection of digitized or born digital items, and your primary goal is to get it online, the easiest thing to do might be to just put it online in a simple directory or two with some web pages listing and describing it all. Make a backup, too, or two, of course. But most of us with web sites can get some new files linked from a library's home page pretty easily these days. If you can do that, your users can find it from your home page, and even if they never visit your home page, the major search engines can find your items and crawl and index them, so maybe your users can find things that way.

If you think this sounds defeatist or desperate, don't think that -- I mean just the opposite. Sometimes the shortest path between users and content is simply putting the content where users have a chance at finding it. The web does this very well for us. I've administered enough web sites and odd software packages over the years to know well that the content that lasts online the longest is almost always the content with the least amount of stuff around it that can go wrong. Whether it's an old programming language or a bad script or an odd toolkit few people ever used, if the only way to get to your stuff is through some oddly-shaped software that's out of date and unreliable, someday nobody's going to be able to get to your stuff anymore. If, on the other hand, the same content is simply available over the web like any other web content, just a bunch of files in a directory on a web server, then that content can be indexed and reindexed by Google et al., copied and recopied onto different servers by you and your staff, and migrated across changes in web server and operating system software choices over time.

We are the repository

The second mistake to avoid is forgetting that we are the repository. Every software repository I've helped to build has faced complex issues of planning and policy which had little to do with technology and everything to do with how to build a sustainable program for ensuring access over time. Remove any special software from the equation completely -- like in the case of "just a few directories on a web server" -- and planning and policy issues still come into play. Like with any other materials in a library, it's ultimately

up to those of us working in the library to set, maintain, and uphold policies for collection development, access, maintenance, and retention. If repository technology gets in the way of making policy choices that fit in with your broader institutional mission, something might be wrong.

Optimize for access

A third mistake that's easy to make is to over-think what a "digital object" might be. I fall into this trap all the time, even with a few rounds of experience under my belt. If you focus on making some content available, using one specialized tool or another or none at all, and that content is useful to your community, its users will tell you how they want to use it. This is another concept echoed both in the "release early and release often" mantra of free software development and the "don't make me think" school of usability testing. Real feedback from real users will tell you the most about features your repository should add or improve, or when one degree more or less of descriptive metadata will make items easier to find or will cost you a ton without really helping people.

If, before "just giving it to people", you spend months or years designing specifications for "complex objects" and hammering structural metadata and content files into shape to match before ever giving your users a chance to see that content, you might just find that you've spent a lot of time and energy without knowing a bit about what people want to do with your stuff. It's possible that you'll guess correctly about what to call files and how to relate files and metadata to each other and store all of that on disk, but that might not help your users at all. Give three programmers a pile of files and ask them to arrange it and you'll get five system designs for how to do it in four different programming languages and database models, but none of that tells you whether your users will use any of it.

The best thing about letting users drive what you do, when it works (I know how hard it can be just to get feedback sometimes), is that it lets you build incrementally. Maybe you start with a simple set of collections published in a few directories and not a lot more. If your users are happy

with that, maybe you can stop there. But if they ask for the ability to search across it all, or to browse it all by subjects, or for specialized functions like being able to zoom in on large images, for instance, maybe that's a reason to look deeper at a specialized software package to augment or replace your "files on disk" setup. If you try a new package, look to your user community to tell you whether it does the job better.

I know this advice isn't going to "solve your repository problem" for you. But if you avoid over-thinking software choices and you avoid over-thinking the complexity of your content, you might find that there are immediate, cost-effective choices available to you that can help your community soon and teach you a lot along the way. And if you focus on optimizing access to meet your community's needs, you might find that the policies you'll need to sustain digital materials over time might match how you do everything else already. In five, ten, and twenty years, after all, any software you use today is likely to be obsolete, but it'll still be your responsibility to make and keep your content available and useful to your community.