From Our Readers: Virtues and Values in Digital Library Architecture

Mark Cyzyk

Editor's Note: "From Our Readers" will be an occasional feature, highlighting ITAL *readers' letters and commentaries on timely issues.*

At the Fall 2007 Coalition for Networked Information (CNI) conference in Washington, D.C., I presented "A Survey and Evaluation of Open-Source Electronic Publishing Systems." Toward the end of my presentation was a slide enumerating some of the things I had personally learned as a Web application architect during my review of the systems under consideration:

- Platform independence should not be neglected.
- One inherits the flaws of external libraries and frameworks. Choose with care.
- Installation procedures must be simple and flawless.
- Don't wake the SysAdmin with "Slap a GUI on that XML!"—and push application administration out, as much as possible, to select users.
- Documentation must be concise, complete, and comprehensive. "I can't guess what you're thinking."

Initially, these were just notes I thought might be useful to others, figuring it's typically helpful to share experiences, especially at international conferences. But as I now look at those maxims, it occurs to me that when abstracted further they point in the direction of more general concepts and traits—concepts and traits that accurately describe us and the products of our labor if we are successful, and prescribe to us the concepts and traits we need to understand and adopt if we are not. In short, peering into each maxim, I can begin to make out some of the virtues and values that underlie, or should underlie, the design and architecture of our digital library systems.

Freedom and equality

Platform independence should not be neglected.

"Even though this application is written in platform-independent PHP, the documentation says it must be run on either Red Hat or SuSE, or maybe it will run on Solaris too, but we don't have any of these here."

Mark Cyzyk (mcyzyk@jhu.edu) is the Scholarly Communication Architect, Library Digital Programs Group, Sheridan Libraries, Johns Hopkins University in Baltimore.

While I no doubt will be heartily flamed for suggesting that Microsoft has done more to democratize computing than any other single company, I nevertheless feel the need to point out that, for many of us, Windows server operating systems and our responsibility for administering them Way Back When provided the impetus for adding our swipe-card barcodes to the ACL of the Data Center—surely a badge of membership in the Club of Enterprise IT if ever there was one. You may not like the way Windows does things. You may not like the way Microsoft plays with the other boys. But to act like they don't exist is nothing more than foolish burying one's head in the *NIX sand.

Windows servers have proven themselves time and again as being affordable, easily managed, dependable, and, yes, secure workhorses. Windows is the Ford pickup truck of the server world, and while that pickup will some day inevitably suffer a blowout of its twenty-year-old head gasket (and will therefore be respectfully relegated to that place where all dearly departed trucks go), it's been a long and good run. We should recognize and appreciate this. Windows clearly has a place in the data center, sitting quietly humming alongside its Unix and Linux brothers.

I imagine that it actually takes some effort to produce platform-dependent applications using platform-independent languages and frameworks. Such effort should be put toward other things.

Keep it pure. And by that I mean, keep it platform independent. Freedom to choose and presumed equality among the server-side OSes should reign.

Responsibility and good sense

One inherits the flaws of external libraries and frameworks. Choose with care.

So you've installed the OS, you've installed and configured the specified Web server, you've installed and configured the application platform, you've downloaded and compiled the source, yet there remains a long list of external libraries to install and configure. One by one you install them. Suddenly, when you get to Library Number 16 you hit a snag. It won't install. It requires a previous version of Library Number 7, and multiple versions of Library Number 7 can't be installed at the same time on the same box. Worse yet, as you take a break to read some more of the documentation, it sure looks like required Library Number 19 is dependent on the current version of Library Number 7 and won't work with any previous version. And could it be that Library Number 21 is dependent on Library Number 20, which is dependent on Library Number 23, which is dependent on—yikes—Library Number 21?

All things come full circle.

But let's suppose you've worked out all of these dependencies, you've figured out the single, secret Order in which they must install, you've done it, and it looks like it's working!

Yet, when you go to boot up the Web service, suddenly there are errors all over the place, a fearsome crashing and burning that makes you want to go home and take a nap. Something in your configuration is wrong? Something in the way your configuration is interacting with an external library is wrong? You search the logs. You gather the relevant messages. They don't make a lot of sense. Now what to do?

You search the lists, you search the wikis to no avail, and finally, in desperation, you e-mail the developers.

"But that's a problem with Library X, not with our application."

Au contraire.

I would like to strongly suggest a Copernican revolution in how we think about such situations. While it's obvious that the developers of the libraries themselves are responsible for developing and maintaining them, I'd like to suggest that this does not relieve you, the developer of a system that relies on their software, from responsibility for its bugs and peculiar configuration problems. I'd like to suggest that, far from pushing responsibility in the case mentioned above out to the developers of the malfunctioning external library, that you, in choosing that library in the first place, have now inherited responsibility for it.

Even if you don't believe in this notion of inheritance, if you would please at least act as if it were true, we'd all be in a better place. Part of accepting this kind of responsibility is you then acting as a conduit through which we poor implementers learn the true nature of the problem and any solutions or temporary workarounds we may apply so that we can get your system up and running pronto. In the end, it's all about your system. Your system as a whole is only as strong as the weakest link in its chain of dependencies.

Simplicity and Perfection

Installation procedures must be simple and flawless.

It goes without saying that if we can't install your system we *a fortiori* can't adopt it for use in our organization.

I remember once having such a difficult time trying to get a system up and running that I almost gave up. I tried first to get it running against Apache 1.4, then against Apache 2.0. I had multiple interactions with the developers. I banged my head against the wall of that system for days in frustration. The documentation was of little help. It seemed to be more part of an internal documentation project, a way for the developers to communicate among themselves, than to inform outsiders like me about their system. And related to this I remember driving to work during this time listening to a report on NPR about the famous Hopkins pediatric neurosurgeon, Dr. Ben Carson. Apparently, earlier in the week he had separated the brains of Siamese twins and the twins were now doing fine, recuperating. The NPR commentator marveled at the intricacy of the operation and at the fact that the whole thing took, I believe, five hours.

"Five hours? FIVE HOURS?!" I exclaimed while barreling down the highway in my vintage 1988 Ford Ranger pickup (head gasket mostly sealed tight, no compression leakage). "I can't get this system at work installed in FIVE DAYS!"

Our goal as system architects needs to be that we provide to our users simple and flawless installation procedures so that our systems can, on average, be installed and configured in equal or less time than it takes to perform major brain surgery.[1]

"All in an Afternoon" should become our motto.

I am happy to find that there are useful and easy to use package managers, e.g., Yum and Synaptic, for doing such things on various Linux distributions. Windows has long had solid and sophisticated installation utilities. Tomcat supports drop-in-place WAR files. When possible and appropriate, we need to use them.

Justice and E-Z Livin

Don't wake the SysAdmin with "Slap a GUI on that XML!"—and push application administration out, as much as possible, to select users.

I remember reading Plato's *Republic* as an undergraduate and the feeling of being let down when the climax of the whole thing was a definition in which "justice" simply is each man serving his proper place in society and not transgressing the boundaries of his role.

"That's it?" I thought. "So you have this rigidly hierarchical society and each person in it knows his role and knows in which slot his role fits—and keeping to this is 'justice'?"

This may not be such a great way to structure a society, but now that I think about it, it's a great way to structure a computer application. Sit down and carefully look at the functions your program will provide. Then create a small set of user roles to which these functions will be carefully mapped. In the end you will have a hierarchical structure of roles and functions that should look perfectly simple and rational when drawn on a piece of paper.

And while the Superuser role should have power over

all and access to all functions in the application, the list of functions that he alone has access to should be small, i.e., the actual work of the Superuser should be minimized as much as possible by making sure that most functions are delegated to the members of other, appropriate, proper user roles.

Doing this happily results in what I call the State of E-Z Livin: The last thing you want is for users to constantly be calling you with data issues to fix. You therefore will model management of the data—all of it—and the configuration of the application itself—most of it—directly into the architecture of the application, provide users the GUIs they need to configure and manage things themselves, and push as much functionality as you can out to them where it belongs. Let them click their respective ways to happiness and computing goodness. You build the tool, they use it, and you retire back to the land of E-Z Livin.

Users are assigned to their roles, and all roles are in their proper places. Application architecture justice is achieved.

Clarity and wholeness

Documentation must be concise, complete, and comprehensive. "I can't guess what you're thinking."

As system developers we've probably all had the magical experience of a Mind Meld with a fellow developer when working intensively on a project. I have had this experience with two other developers, separately, at different stages of my career. (One of them, in fact, used to point out to everyone that, "between the two of us, we make one good developer!") This is a wonderful and magical and productive working relationship in which to be, and it needs to be recognized, supported, and exploited whenever it happens. You are lucky if you find yourself designing and developing a system and your counterpart is reading your mind and finishing your sentences.

However, just as it's best to leave that nice young couple cuddling in the corner booth alone, so too it really doesn't make a lot of sense to expect the Mind-Melded developers to turn out anything that remotely resembles coherent and understandable documentation. Those undergoing a Mind Meld by definition know perfectly well what they mean. To the rest of us it just feels like we missed a memo.

If you have the luxury, make sure that the one writing the documentation is not currently undergoing a Mind Meld with anyone else on the development team. Scotty typically stayed behind while he beamed the others down.

Beam them down. Be that Scotty. You do the world a great service by staying behind on the ship and dutifully reporting, clearly and comprehensively, what's happening down on the Red Planet.

To these five maxims, and their corresponding virtues, I would add one more set, one upon which the others rely:

Empathy and graciousness

You are not your audience.

At least in applied computing fields like ours, we need to break with the long-held "Guru in the Basement" mentality. The actions of various managerial strata have now ostensibly acknowledged for us that technical expertise, especially in applied fields, is a commodity, i.e., it can be bought. A dearth of such expertise is remedied by simply applying money to the situation—admittedly difficult to do at the majority of institutions of higher education, but a common occurrence at the wealthiest. Nevertheless, the dogmatic hold of the Guru has been broken and the magical aura that once draped her is not now so resplendent—her relative rarity, and the clubby superiority that depended upon it, has been diluted significantly by the sheer number of counterparts who can and will gleefully fill her function. We respect, value, and admire her; it's just that her stranglehold on things has (rightfully) been broken. And while nobody is truly indispensable, what is more difficult and rare to find is someone who has the Guru's same level of technical chops coupled with a genuine empathic ability to relate to those who are the intended users of her systems and services.

Unless your systems and services are geared primarily toward other developers, programmers, and architects—and presumably they are not, nor, in the library world, should they be—your users will typically be significantly unlike you.

Let me repeat that: Your users are not like you.

Rephrased: You are not your audience.

When looking back over the other maxims, values, and virtues mentioned in this essay then, the moral-psychological glue that binds them all is composed of empathy for our users—faculty, students, librarians, non-technical staff—and the graciousness to design and carry out a project plan in a spirit of openness, caring, flexibility, humility, respect, and collaboration. When empathy for the users of our systems is absent—and there are cases where you can actually see this in the design and documentation of the system itself—our systems will ultimately not be used. When the spirit of graciousness is broken, men become robots, mere rule followers, and users will boycott using their systems and will look else-

where, naturally preferring to avoid playing the Simon-Says games so often demanded by Tech Folk in their workaday worlds; there is a reason the comic strip Dilbert is so funny and rings so true. When confronted with a lack of empathy and graciousness on our part, the users who can boycott using our systems and services will boycott using our systems and services. And we'll be left out in the rain, feeling like, as Bonnie Raitt once sadly sang, "I can't make you love me if you don't / I can't make your heart feel something it won't." Empathy and graciousness, while not guaranteeing enthusiastic adoption of our systems and services, are a necessary precondition for users even countenancing participation.

There are undoubtedly other virtues and values that can usefully be expounded in the context of digital library architecture—consistency, coherence, and elegance immediately come to mind—and I could go on and on analyzing the various maxims surrounding these that bubble up through the stack of consciousness during the course of the day. Yet doing so would conflict with another virtue I think is key to the success and enjoyment of opinion-piece essays like this and maybe even of other sorts of publications and presentations:

Brevity.

Note

1. A colleague of mine has since informed me that Carson's operation took twenty-five hours, not five. Nevertheless, my admonition here still holds. When installation and configuration of our systems are taking longer, significantly longer, than it takes to perform major brain surgery, surely there is something amiss?