Does free software make sense for your enterprise?

Finding free software at your office is like finding a Republican in San Francisco

Tom Jackiewicz

"Dude, I can, like, totally do that way cheaper with Linux and stuff."

These were the words of a bearded geek running Linux on his digital watch. As he proceeded to cut and patch alpha code into the Linux kernel running on the production database system, the manager watched on in admiration. Six months later, long after the young hacker decided to move into a commune in the Santa Cruz hills, something broke. Was it really "way" cheaper?

Nostalgia and first impressions

I remember the first time I actually opened up a Sun server after years of using them remotely. Coming from a background of PC-based hardware (that were somehow deemed servers), it was a memorable moment. Some obsessive-compulsive engineer with an eye for detail had obviously spent countless hours making sure that the 2-inch gap between a hard drive and an interface port was filled with (gasp!) a 2-inch cable. Each wire was color-coded, the appropriate length, and carefully held down with ties and widgets to ensure that they never got in the way. Each server I opened was configured the exact same way, and each component matched—down to the screws (which were custom fitted and just right). This was a far cry from 9 foot 2-device IDE cable I got out of the junk drawer that was used to add a random hard drive held in place by duct tape. As the halo above the server lit up the room, I was suddenly struck with a justification for the hefty price tag for these magical machines. Quality.

The SPARCstation 20, as beautiful as a rose

As the halo above the server lit up the room, I was suddenly struck with a justification for the hefty price tag for these magical machines

The same quality control that went into the server packing went into the hardware. Sure, I couldn't connect my Colorado tape drive, digital camera, or latest toy to the back of these servers as I could to the PC, or use a really cheap hard drive created by some no-name vendor, but all the supported hardware that I actually did connect was controlled by a perfect driver and was carefully integrated into the operating system. It all just worked. Magically. If it was on the support list, you knew that some team of detail-oriented engineers took care to make sure that there were no flaws.

Someone had to pay for all of this, hence the hefty price tags, but I knew that most of the servers that I was running

my mission critical applications on were deployed when I was in diapers. Baby diapers, not the diapers that people who remember this type of quality are wearing today—or the ones I'll be wearing when these servers finally crash.

The alternatives to Sun Microsystems, IBM, Solaris, AIX, HP, HP/UX and all of the other commercial software running on proprietary hardware were untested. Linux was in its infancy and focused on desktops and attempted to support everything you plugged into it. It was a time of instant gratification and first to market that was necessary for it to gain acceptance in the desktop world. If a digital camera worked in Windows, the Linux community had better jump on providing support for it in their world. This led to terrible drives, kernels with goto statements and functions with the comments "Uhm, we'll get to this later. Return 0 for now". This led to instability when these systems were used as servers. The BSD community, while providing more stable software, wasn't seen as sexy and didn't gain as much acceptance. FreeBSD, NetBSD and OpenBSD were "theoretical" operating systems that were coded well but weren't supported enough to provide integration with some of the more common components in use in current infrastructures. Additionally, more focus within the BSD community was spent on ensuring software was functional (and up to standards) than pretty—which led to a lack of usable interfaces. This gap seemed to propel commercial software more and more. They were well programmed and the vendors had enough resources to cover usability and stability.

When the engineers, system administrators, programmers, and jack-of-all-trades geeks move and finally become managers (or are forced into the role), they remember these days. And they associate Sun Microsystems, IBM, and other giants with this type of quality. To them, the quality they first saw and admired is still around today. Decisions on what to use is made by these new managers.

Who runs this thing?

Use of free software and alternatives to expensive proprietary hardware went crazy during the early days of the new AOL—err internet. Instead of the goal being stability and an infinite life, new companies were satisfied with getting something out quickly and, upon gaining venture capital or selling out, they could "move up". The investment required for the commercial side was just too much for a fledgling company. But as we all know, a temporary solution usually becomes a permanent part of your infrastructure. Even worse, a temporary solution in place until more funding is available will outlast the company CEO.

But as we all know, a temporary solution usually becomes a permanent part of your infrastructure. Even worse, a temporary solution in place until more funding is available will outlast the company CEO

Adding to the equation, the open source and the free software movements were growing and the quality of the software was definitely increasing, providing a reasonable solution that wouldn't instantly crash.

Unfortunately, the problem facing management was responsibility and support. If there was a problem with software written by a 16 year old kid in Finland, who was responsible. If an administrator walked in and deployed a quick solution to fix your immediate needs, who would support his undocumented idea when he left? As employees leave and are no longer expect to grow up with the company (especially during the years of high school age CEOs) as they once have been. A need for a redundant array of inexpensive administrators is created. The prerequisites for this need are an infrastructure that is supportable by many.

Your free software based system running alpha drivers for that new array to save you 500 bucks? Gone, you're the only one who can run it. Your sendmail infrastructure optimized to take advantage of inferior hardware using custom interfaces for account management? Gone, welcome the appliances.

The need to have software and hardware maintainable by anyone with a base level of experience has replaced making the most of the hardware and software in your infrastructure. If the configurations aren't default, the performance improvement that you might be giving them with your take on things just won't be cost effective. Look at it like blackmail or extortion. You walk into a company, "save" them hundreds of thousands of dollars on software, get them locked into your infrastructure, and then demand a huge raise just to keep it going because no one else can. By the time they've moved their operations towards your in-

frastructure, they can't easily go back. Ironically, the same came be said for commercial software—even those based on open standards. That one extra little feature that Microsoft has in their implementation of a product over the free software version will lock your company into using Microsoft products for all eternity. At the same time, your company will feel that they could easily move away and use any vendor before, hey, it's an open standard!

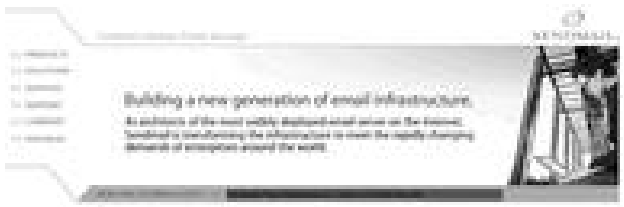
Consistency in project life cycles

Ok, so I used a buzzword, but my head isn't standing up into point. This matters. There are processes in place for development of software, quality assurance testing, and validation that happen before software reaches the customer. In the commercial realm, there are people paid to do some of the most boring tasks just to make sure they get it right. While 30 QA engineers aren't necessarily going to be as good as a public user community, they are consistent in their operations and try to make sure that nothing slips through the cracks. The user community will often test a few things here and there but won't go through the tedious tasks of making sure some of the most basic operations work with each subminor revision of code. If something is missed, the assumption is that someone else will find it and any potential problems will be taken care of.

These things are boring. But someone has to do it. From the programming, QA, and other areas of software, each has a defined process that needs to be followed. The same is true for deployment within your infrastructure. What would you do to bring Postfix, which is amazing software, into the mix within your environment? Most people would take some of the old email infrastructure, validate a few things here and there (to make sure no users requiring the delivery of mail are missed). An old legacy host doesn't speak well with your email gateways? Uh oh, you overlooked that. Important emails being inappropriately tagged as spam? My bad, sorry. These mistakes happen, and because these little things were overlooked in your haste to show off Postfix, someone is going to look bad.

Try deploying any commercial package (especially with a support contract). All of the little caveats you run into will surely be documented by the vendor. A list of supported products will also be given so you know which integration

Have a problem? Blame sendmail.com, the commercial side



efforts will require a bit of extra work or should remain pointed to an interim solution. And if all else fails…

Who's to blame?

We're a society of lawsuits and passing the buck. Slip and fall while running the wrong way on an escalator drunk? First thing someone will say is that there wasn't a sign telling you not to do that. Sue someone for your own stupidity. Fall behind on a project and lose your bongus because of a bug in software from a vendor? The threat of tarnishing their reputation lets you strong arm them into giving you anything you want. Big or small, there's someone on the other end of the software with a livelihood.

The threat of tarnishing their reputation lets you strong arm them into giving you anything you want. Big or small, there's someone on the other end of the software with a livelihood

The only person to blame when free software fails is the person who deployed the software. But the person in charge of your organization can't just say "Oh, that crazy Ted! He did it again!" and get away with it. Heads roll when something bad happens. If the bad thing is free software, the heads are internal. If the bad thing can be blamed on a vendor then more people within your organization are safe from the axe. Companies all like it when there's someone outside of the organization that can be blamed for a failure.

Sun and other large vendors have teams of highly trained engineers ready to parachute into your company at a moment's notice. All are trained consistently, all read from the same handbook, and all can come in and give you 2080 hours of

Check out sendmail.org, the original free version of the software

work in a weekend just to get you up and running. Try doing that by bringing in the same people "trained" on free software. Each has their own idea on how to do things, no consistent sets of manuals, and, for the most part, all from different companies. There isn't a single source of gurus who know the same Linux implementation who can run out to help you when you're in a bind.

At the same time, a list of truly supported packages will be there. If you try to integrate a commercial package with something that isn't supported, there are no guarantees. There's no "It should work" here. These certifications by the vendor are often done after extensive testing from both sides—the package being deployed and the package being integrated. These often leave you with an all-commercial environment as no one is going to have the time or money to make sure that something is going to work with the latest free software version of something. These things leave free software back a bit but make your management rest a little easier at night. After all, if Microsoft certifies something will work, it will, right?

If you can't beat 'em...

The open source and the free software communities saw some of the same issues that I just ranted about on my proprietary soap box. Their response? Sendmail, so established a software that is has been the victim of a love-to-hate mentality, never had much competition in the commercial realm for much of its life. Eric Allman (father of sendmail) could have chosen to sit quiet and not do anything to further sendmail. Instead, he chose to create a commercial version of the software, offer support, and create an alternative to commercial email packages that were up and coming but not yet a threat. The end result was the same sendmail everyone was already running with the added aspect of accountability and a helping hand. This ended up being a good move because when the dot com boom happened, the former telemarketer turned "engineer" didn't understand the new set of simpler m4 configuration macros used within sendmail, let alone the ability to understand anything required more than 3 mouse clicks and a "Duh!"

Apache, like Sendmail, has always had a stable following. It lived through commercial pressure (though one can always question the legitimacy) of Netscape Commerce and Enterprise Servers, iPlanet, IIS, and a slew of poor quality commercial versions of their software. A foundation was started and an entire community grew to support each other and their relevant projects. While there wasn't necessarily someone on the other end of a phone line, there was an established group, which your managers would have a hard time convincing you, would magically disappear.

Sendmail and Apache moved away from the unorganized efforts seen by the free software community when it was smaller and less significant. This gave them credibility and made some of their software a viable replacement for commercial products.

What can be done?

What can you do about all of this, if anything? Don't be the bearded geek I mentioned at the beginning of the article and put in some extra effort to make sure that what you deploy is supportable. Baby steps. You won't win anyone over right away but when you deploy a free replacement for software, document the process, live by standards, and make sure that the life cycle of your project involves a significant amount of quality assurance, integration and interoperability testing. Put in the extra time to see what sorts of processes exist within your company and what will make everyone comfortable. Take the extra effort required to follow these

Check out apache.org, home of quality projects near you

procedures. Not only will you earn respect, you might learn something, come up with a bug, and let management view your free software deployment in a better light. Don't just "do it" and expect everything to work magically with your 3am software install.

The community as a whole should also spend time making sure that their software meets the requirements set by other vendors for interoperability. Software from Sun and Microsoft have a strict set of guidelines before they'll list a product as fully supported. While expensive, going through the process of certifying free software as fully operable with commercial packages will give a big boost to the movement.

One problem area that people fall into is the latest and greatest upgrade cycle. The stability of a system goes down if you keep upgrading your kernel, libraries, and applications to the latest and greatest revision. There's really no reason to fix something that isn't broken. In many environments, constantly upgrading libraries will cause components to fail. It will also cause custom code written by programmers in your company to act different. The lack of backwards compatibility in some packages just doesn't work in a production environment. Sure, these problems exist in the commercial realm as well (and the hiccups are even worse) but those systems aren't upgraded as often. A patch upgrade every 6 months or a year (with minor security updates in between) aren't going to create as big a problem as a weekly kernel upgrade just for the sake of a kernel upgrade. It might be boring to sit around and wait while your office servers fall a few revisions back from your desktop at home, but it's worth it. Schedule significant system upgrades after testing in development environments (you know what those are, right?) for a while. Create integration environments so that developers and application groups can make sure their code is going to work with your proposed update. Stay behind the curve a little bit and let others find out bugs in the new code before you're running it on a system that must be up 99.9999% of the time.

Discipline, process, and doing a few extra tedious tasks will give everyone a better impression of some of the solutions you're proposing. Maybe, just maybe, quality software will catch up with the vendors. Maybe the smiles on their faces won't be so big and their thinning bank accounts will make them realize that we should all work together to create better code and worry more about things that matter—not just the bottom line.

Bibliography

Jackiewicz, Tom "Deploying OpenLDAP", Apress:2004

Copyright information

About the author

Tom Jackiewicz is currently responsible for global LDAP and email architecture at a Fortune 100 company. Over the past 12 years, he worked on the email and LDAP capabilities of the Palm VII, helped architect many large scale ISPs servicing millions of active email users, and audited security for many Fortune 500 companies. Jackiewicz has held management, engineering, and consulting positions at Applied Materials, Motorola, and Winstar GoodNet. Jackiewicz has also published articles on network security and monitoring, IT infrastructure, Solaris, Linux, DNS, LDAP, and LDAP security. He lives in San Francisco's Mission neighborhood, where he relies on public transportation and a bicycle to get himself to the office-fashionably late. He is the author of Deploying OpenLDAP, published by Apress in November 2004.