



DAVID ASCHER, ACTIVESTATE

Is Open Source Right for You

A fictional case study of open source
in a commercial software shop

The media often present open source software as a direct competitor to commercial software. This depiction, usually pitting David (Linux) against Goliath (Microsoft), makes for fun reading in the weekend paper. However, it mostly misses the point of what open source means to a development organization.

In this article, I use the experiences of GizmoSoft (a fictitious software company) to present some perspectives on the impact of open source software usage in a software development shop. My intention is to help managers, architects, and developers answer these questions:

- Why should we use open source software?
- When should we use open source software?
- How should we use open source software?
- How will using open source software affect the way we do our jobs?

Let's walk through the halls of GizmoSoft and find out what people are saying about the company's use of open source software. (Note: the events and characters depicted below are fictional and do not represent any person living or dead. The gossip is also fictional.)

IN THE ELEVATOR: THE STATEMENT OF CORPORATE INTENTIONS

Joanna (director of marketing): You know, for once I enjoyed our staff meeting. I'm glad that Joe [the CEO] is willing to make a public statement about how our software development practices are better than the competition's. From a marketing point of view, it's nice to be able to leverage the buzz around Linux. It really differentiates us from the rest of the crowd. It's been hard making "Web services support" sexy when Web services



Is Open Source Right for You?

are sooooo boring to pretty much everyone I talk to.

Francis (in-house lawyer): Yeah. That reminds me, I need to talk to Susan [the director of development]. We've got to work out the legal stuff. I'm really leery of the licensing issues.

Joanna is happy because the CEO announced that the board has decided to "go public" with an internal initiative to aggressively use open source in the company's software development practices. Privately, the board is hoping that it can cut down development costs and make the company more appealing as an acquisition target.

Joanna is seeing the press jump on open source because of the "whole Linux thing," and she thinks the company can benefit from riding on that bandwagon. Francis, on the other hand, is in a bad mood because of the same staff meeting. He has seen some open source software licenses, and he doesn't like them one bit. They're fuzzy, written by idealists with no concept of the law, and they make him very nervous. In the past, he has dealt mostly with contract and licensing implications where the intellectual property ownership was clear; now suddenly he needs to educate the director of development on the risks of dodgy licensing. No one seems to realize that it's his job on the line if the risk isn't managed right!

IN THE DEVELOPMENT BAY: IMPLEMENTING OPEN SOURCE

This is the hub of development activity. There are eight developers working on two projects. One project is in the early planning stages, and folks are having fun. The other project is two weeks late, and the massage therapist around the corner is making good money from that team.

Stu has been with the company only two months—he's the youngest developer on staff, and life is good. He has been assigned to build a directory function into the next revision of one of GizmoSoft's main products. Stu recently graduated from college, and he's all pumped up about building some *real* software, not the toy projects he worked on in school. Over the weekend, he decided that the right thing to do is to use OpenDir. After all, he knows it, he's seen it work at school, and he'd love to get involved in an open source project for work. That would rock.

The problem is that Stu's mentor, John, has another

solution in mind, based on code he wrote years ago, called BorkDir. It's time for the weekly status meeting:

John: How are you doing on figuring out what directory system to use?

Stu: Good! I think that OpenDir will work just fine, like I said last time.

John: Hmm. I was looking at their release schedule. They seem to release software all the time—something like 50 releases in the last seven years. That doesn't strike me as being really solid software.

Stu: Uh, well, you know, most of those are, like, patch releases, you know?

John: I guess. What's wrong with using BorkDir? It's not that I want my code to be used at all costs, but why not use our code if it's good enough?

Stu: I don't know what's wrong with BorkDir. You were away last week, and I couldn't find anyone else in the company to tell me how it works.

John: Ah, right. Are you sure that OpenDir will do everything we need?

Stu: Pretty much. BeHeMothCo.com uses it for their system, so I can't imagine that it wouldn't work for us.

John: OK, fine, let's give it a shot.

This conversation highlights some issues that are worthy of commentary.

Stu, fresh from the collaborative environment he had in college, is eager to join in the "open source movement." John is more reluctant. His reluctance results from his discomfort with a foreign way of working, much too public and communal for his taste. Also, John is concerned about software quality. To Stu, the frequent releases indicate dynamic development; to John, they indicate a failure of process.

Both Stu and John are partially correct. It's true that open source projects tend to evolve processes (release definitions, release management, quality control, and the like) late in their life cycles (or, sometimes, not at all). On the other hand, the metrics by which a "reasonable release rate" is defined differ between commercial and open source software. Commercial software has to contend with marketing costs, increased customer support, and delayed revenue when a new release comes out. Open source projects—especially when supported by a management infrastructure such as SourceForge.net—can produce minor or micro releases comparatively easily.

If you decide to use an open source package in a commercial product, plan to evaluate each release of the package to understand which ones are "worth" integrating. These evaluations need to be made at appropriate times in the software development process (when the risk can

be tolerated) and should include a variety of criteria, such as: whether bugs important to *your* project got fixed; what the expected developer and QA (quality assurance) effort involved in “upgrading” is; and what the risk implications are for the system as a whole, given the nature of the changes and the testing and QA schedule, etc. With the right framework for making the decision in place, evaluating frequent releases becomes, at worst, a minor issue; and in the case of very frequent releases, these evaluations can be batched.

The release frequency, however, is likely a red herring in this case—the comparison between OpenDir and BorkDir isn’t just between an open source package and a proprietary one. It’s also a comparison between a popular open source project and a component built in-house by the person who will decide which package to use. This creates an emotionally charged situation for John, who is likely to suffer from at least a temporary bout of NIH syndrome (whereby things Not Invented Here are deemed inherently worse).

However, John has considerable experience with the “problem” (building a directory function for the GizmoSoft product) and, more important, with the overall requirements of the system being built. Ignoring his experience would be a serious failure. A good approach would be for John to analyze the open source package in concert with Stu. This would bring John’s experience to bear on the problem (as many open source projects are not applicable in enterprise settings simply because they haven’t grown up in those environments). Also, this process would teach Stu how to evaluate software critically, a task that is likely to become more important in the future.

Such an evaluation might reveal that BorkDir has crucial features that the open source equivalent lacks. Overall, the open source project might be more complete and certainly more widely tested than in-house code (which has probably been used previously in only one application, if at all). In that case, John and Stu should evaluate the effort required to add the critical features to the open source project. Then they must make the difficult choice of whether to contribute the features back to the open source project or keep it private. (This will have implications for their future use of the code, as described in a later section.)

IN THE WAR ROOM: OPEN SOURCE IN ACTION

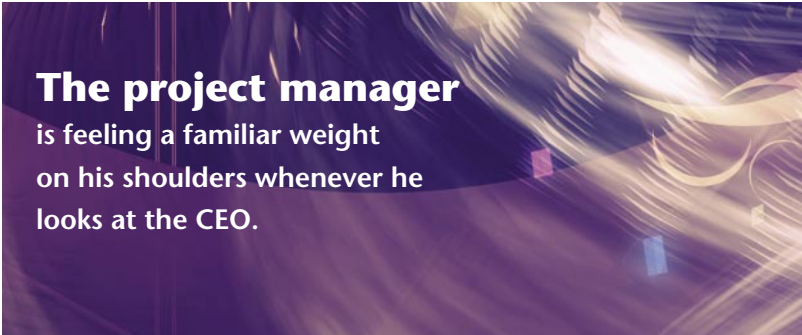
The project is late. The project manager is feeling a familiar weight on his shoulders whenever he looks at the CEO. The CEO is trying hard not to ask every day, “Is it ready yet?” He doesn’t really need to; his body language

says it for him.

Now it’s time to review last night’s build to see if it’s ready for release. There are the usual problems with regressions, new bugs, performance problems, and installation snafus that all need to be prioritized, fixed, or documented. But today there’s a new and unexpected twist in the “launch decision” status meeting: a critical bug in an open source component of the product.

Sam (developer): Uh, something came up late last night that I think we should discuss. The bug that we thought was in our code was actually traced down to the widget itself, which you remember is provided by the MyToolkit open source library. Does anyone here know that code? <silence>

Sandy (project manager): Who owns that code? <silence and head-shaking>



The project manager
is feeling a familiar weight
on his shoulders whenever he
looks at the CEO.

Sandy: Uh-oh. Who checked it in?

Tom (lead architect): I think Stacey did, six months ago, back when she was working on the front end. I don’t know if anyone’s kept up with the MyToolkit project. It’s possible that the problem has been fixed. [Tom checks the project on SourceForge with his new tablet PC, just to show off.] Yeah, well, the version we have in our code is 2.14.a, and the current release is 3.0.4. There are also more versions in the 2.x branch. I have no idea whether our problem has been fixed yet, or, if so, in what version. Sandy: All right then. Tom, can you work with Sam to answer both of those questions, and give us a status report by end of day? This is top priority, folks, since the bug is a blocker, and as far as I can tell, we have *no idea* how long it’ll take to fix. The QA group will presumably need to redo a full pass on the user interface after the fix as well. <sighs>

Late releases are bad enough; in this case, the fact that the blocker bug resides in open source code highlights some common failures in the development process:

1. Because the software component is written outside the building, it hasn’t been monitored as thoroughly



Is Open Source Right for You?

as software written by GizmoSoft employees. If a component is important enough to block a release, it must be monitored on a regular basis. The fact that several MyToolkit releases (which probably addressed numerous documented bugs) were “missed” indicates that no one thought that GizmoSoft software would inherit these bugs. As discussed earlier, upgrades to open source components must be considered as part of GizmoSoft’s release planning. Note to open source authors: Open source projects with up-to-date unit tests are much more likely to be adopted by big companies, since “customers” can compare the unit tests between releases to see what has actually changed. This is better than trusting documentation, which is often lagging or incomplete.

2. Key developers for the MyToolkit software do exist; they just happen not to work for GizmoSoft. Tom and Sam must find out whether upgrading the MyToolkit component will fix the problem, but they don’t know the code. It would be much easier if one of the key developers of the MyToolkit project were on speed dial, either through a contractual relationship or just as an advisor. That person can probably answer the question within 30 seconds, and, unlike most commercial software shops, there probably isn’t a firewall of support personnel between the knowledgeable engineers and their phones. If you care about a piece of code, it’s worth building relationships with the key authors of that code, even if they don’t “belong to you.” Note that in cases like this one, you may need to be able to cut a check quickly even if it’s for a relatively trivial amount. One suggestion for companies wishing to use open source efficiently is to set up the equivalent of an office supplies slush fund for software fixes.

IN SUSAN’S OFFICE: THE LEGAL IMPLICATIONS

The following exchange is quite promising. Francis takes his responsibility as legal guardian of the organization seriously enough to anticipate problems that may arise from using open source. He discusses these concerns with Susan, the person who is ultimately responsible for what goes into GizmoSoft’s software. Susan has done her bit, analyzing a set of licenses as best she can, and communicating that analysis to her team. It’s therefore less likely

that the company will unwittingly violate the terms of a license.

Francis (in-house lawyer): Susan, about this open source thing—I hadn’t realized that we are actually planning to ship open source software in our product. I’m a bit concerned about that.

Susan (director of development): Ah. Did you see my e-mail back in November? I discussed the license types that seemed OK for us to include in our software. I’m sure you were cc’ed on that e-mail. I basically broke it down into three categories: projects we can use internally; components we can use in our own products (with my version of the legalese so that developers can understand what they have to do); and code that we shouldn’t touch at all. I can send you that list again if you want.

Francis: Yes, please do. I got a call from a board member who’s quite concerned, what with the SCO lawsuit and all. I hate it when non-lawyers start telling me what legal risks to watch out for. All of a sudden, everyone’s an expert on indemnification and copyright law, except that they have no understanding of basic contract law.

Susan: I know how you feel. Everyone thinks they know the best way to write software, but they’ve never had to ship software on a particular day before. I wish we didn’t have to play lawyer, but, unfortunately, the developers and I end up reading software licenses quite a bit these days. Say, how would you feel about doing a brown bag lunch for the dev team about IP law, licenses, and all that?

Francis: Sounds good. If nothing else, they should know who I am so they can ask me questions when in doubt. How about Friday?

Susan and Francis clearly trust each other to do what’s right for the organization. This can be difficult to achieve, given the philosophical distance between legal wonks (who sometimes look at risk as an evil that must be contained at all costs) and propeller heads (who sometimes look at lack of risk as death-by-boredom and who thrive on using “the new thing”).

Ideally, Susan and Francis would set up processes to streamline open source usage and prevent license violations. For example:

- New developer orientation should include a briefing on licenses and other aspects of intellectual property law, so that new staff members are aware of the situation.
- The project manager’s responsibilities should include license checking, so that proper checks are put in place to avoid, for example, adding open source code with “viral” licensing to a proprietary product. Someone should also analyze the downstream consequences (such as documentation requirements) of including

each software component (or, often, each license type).

IN SAM'S OFFICE: THE SPIN

The CTO (Sam) and the product manager (Susan) are meeting with the lead architect (Tom) of the next-generation product, due out sometime next year but not yet defined. Tom is the person most responsible for getting the CEO to utter the words “open source” at the last staff meeting; he’s getting quite good at pitching ideas to upper management. Today his goal is clear: he has decided (for purely philosophical reasons, but he’ll never admit to that) that the company’s next product should have at least 50 percent open source content. He’s well aware that it’s a silly goal from a business perspective, so instead he’s talking about the other benefits.

Tom: Let’s build it on top of the stuff from Apache. Apache is best known for its Web server, but offers a wide variety of software packages, mostly Java-based. It’s really quite good!

Sam: I don’t get it. We just bought SmallCo last year because it supposedly had good technology in that space. What happened to that plan?

Tom: Have you talked to those guys recently? Since Grisha left, it has become pretty clear that that code is patched together with duct tape and the back end is powered by an ant farm. I think the team is really good, and they’ve been able to do great stuff with the code they had to deal with, but I *really* don’t want to build our new flagship product on top of that foundation.

Sam: <grumble>. Didn’t you do due diligence on that codebase?

Tom: How was I supposed to do that in one week, in the middle of a release?

Susan: Let’s not revisit that, no point in it. Can you tell me more about this Apache stuff?

Tom: Yeah, it’s been in development for a few years, but recently it’s become quite solid, especially since IBM has been involved—it’s really good about being standards-compliant. I’ve also heard rumors that Nokia is involved behind the scenes, which could be good if we finally roll out our mobile strategy.

Sam and Susan simultaneously: Cool!

Susan: I assume that would push out our release date, though, right?

Tom: What? No, no, absolutely not. In fact, I’ve been building a prototype just to get a feel for it, and I’m pretty sure that we could have an alpha out earlier than what we’ve currently got planned.

Susan: Why earlier?

Tom: Because our foundation will be much better tested

than it would have been otherwise. Something else that’s neat is that I plugged in another open source project and got a really nice scripting language built in for free, which should make prototyping go even faster.

One more thing... I’m thinking that we should look in that community for the team leader we’ve been interviewing for. It would be very nice to have someone on staff who knows that code really well, given its importance.

Sam: Sounds right. Make sure you don’t get a rabid anti-capitalist, though.

Tom: Right—I hear IBM and Nokia are hotbeds of rabid anti-capitalism.

Susan: Please get someone who bathes and wears shoes.

Here, Tom has quite astutely showcased some of the potential (and at this stage in the conversation it’s all about potential) benefits of using open source software components as part of a product development plan:

Battle-tested software. Reputable software projects, like many run by the Apache Software Foundation, tend to be battle-tested in many types of environments (in fact, in many more environments than you are likely to have available for testing).

Standards compliance. Especially when dealing with Internet technologies like the Web, Web services, XML, etc., open source projects are often highly standards-compliant. Something about open standards appeals to the pride of open source developers. Open standards set absolute targets for project goals, which are easier to quantify and validate than fuzzy concepts such as “good user experience” or “user-friendly” (where commercial software shops tend to do better).

Visibility into future systems. You may not be developing on embedded devices or bleeding-edge platforms today, but others are. If they buy into the open source model, they will probably contribute their work (patches, documentation, experience) back to the project. Therefore, when your business needs take you to that device or platform, much of the groundwork will have been done for you, something that never happens with homegrown systems.

With these arguments, Tom has appealed to Susan, the product manager, by implying quality, risk reduction, and customizability. He has appealed to Sam, the CTO, by suggesting “easy” ports to new platforms and standards compliance.

IN THE FUTURE—WHITHER GIZMOSOFT?

GizmoSoft will eventually encounter other drawbacks of using open source software. Most of these can be mitigated by proactive planning and management.



Is Open Source Right for You?

No or poor documentation. Many people write open source software for free. Almost no one writes open source documentation for free. Lack of documentation (especially for immature projects) can be remedied in two ways:

1. Read the source. After all, many engineers, after bad experiences with commercial software, will not trust the documentation, anyway, and would much rather extrapolate functionality from the program source.
2. Bring your corporate resources to bear. Open source projects often include people who can write documentation but have no motivation to do so. Money can get core documentation written, which will often be maintained for free by the project on an ongoing basis. This is “seed money” well spent.

The software doesn’t meet your needs. There is no general remedy to this problem, but some analysis can often help. It could be that the problem domain is not well suited for an open source software solution. In those cases, it’s best to realize that early and come up with an alternative plan.

Sometimes, however, the open source project leaders focus on problems that are interesting to themselves, but are willing to refocus on your problem—if they know about it. Open source software developers can gain credibility and enhance their professional reputations when their open source projects are adopted by commercial companies. Therefore, don’t be afraid to talk to the project’s contributors. It’s possible that all the project needs to solve your problem is to know about your problem.

Alternatively, some projects are willing to accommodate sponsored work, whereby you pay for the addition of particular features. Don’t assume, however, that you can share the cost with other companies; that’s seldom practicable.

The software project dies. This can happen when there is too much concentration of responsibility in one or two individuals. To avoid this, it’s best to assess the long-term prospects of a project before adopting it. If the main authors give up on a project, you may have no choice but to take it over yourself.

You’ve used an open source project, made some changes, and didn’t contribute the changes back to the project. Maybe you were worried about protecting your

intellectual property; maybe you were concerned about competitive advantage; maybe you were lazy. The project moved on, and now you have an increasing maintenance load because you want the enhancements that the main team is doing to “the trunk,” but you also want to keep your changes. (In other words, you’ve “forked” the code.) In that case, you should review the wisdom for keeping the changes separate. There is no easy answer here, especially if there is “competitive IP” in your changes, but it’s worth critically assessing the true value of that IP and comparing it with the true cost of ongoing fork maintenance.

There are, in most cases, no 1-800 numbers to call. While some organizations do provide support contracts, it’s certainly not true for most open source projects. In general, your ability to get timely support will be contingent on your ability to make volunteers care about your needs—and you shouldn’t be surprised if you get conflicting answers from ill-informed but eager helpers.

Open source software components, if incorporated carefully, can contribute velocity, robustness, and adaptability to commercial software development. To ensure such a positive outcome, it’s worth understanding what changes an organization should anticipate as part of such an effort. As we’ve seen in this article, the biggest impact isn’t really at the level of source code (after all, the code doesn’t know which license it’s under). Instead, because what makes open source code “open source” is the nature of the *relationships* between the contributors and the *processes* surrounding development, those are the areas in which using open source will impact your organization, at every stage of the development process.

Late breaking update: Rumor has it that the rapid success of GizmoSoft’s new product, coupled with the company’s solid presence in its traditional space, has made it ripe for acquisition by BeHeMothCo.com. It turns out the benefit of open source in this case wasn’t in lower costs but better products faster. ☹

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

DAVID ASCHER is chief technologist for ActiveState, which provides open source programming tools for languages, language distributions, and enterprise support services. A Python book author and a director of the Python Software Foundation, he has extensive technical and business experience integrating open source projects such as Python and Mozilla into commercial products.

© 2004 ACM 1542-7730/04/0500 \$5.00