Essay

Here's how an open-source development model can help commercial software companies overcome many of the challenges they face in growing their business.

Setting Up Shop: The Business of Open-Source Software

Frank Hecker, Netscape

Software companies face many challenges in growing their businesses. Product lines must evolve—new products and add-ons to existing products are essential to bring in new incremental revenue. Product quality must be monitored and improved. Engineering must support current and older releases while still driving innovation. Employees must be motivated by interesting opportunities and more than standard incentives. Many companies must also recruit third-party developers and integrators, who in effect help sell the company's products by increasing their value to customers.

These software business challenges are interconnected in two ways. First, most if not all are functions of constrained resources. Few companies have enough people, money, or time to do everything that needs doing, especially when competing against larger companies with greater resources. Second, a strategy exists to address all these challenges at once: turning some (or in exceptional cases all) of a company's software products into open-source ones.

0740-7459/99/$10.00 © 1999 IEEE

Making a product open-source means making the source code for that product freely available under liberal licensing terms, and with no licensing fees. Others are free to take that software, make changes to it, and use or distribute the resulting modified versions as they see fit. When a company makes the right products open-source and chooses an appropriate business model, it can ultimately benefit in ways that more than offset any short-term loss of profits that might stem from no longer being able to sell those products in the traditional way.

A company can choose to make source code freely available and still serve its own business interests as a for-profit organization.

Eric Raymond's article "The Cathedral and the Bazaar" (http://www.tuxedo.org/~esr/writings/cathedral-bazaar) makes a case for open-source development within an extended developer community as a way to create better software. My goal in this article (a full-length version is available on the Web at http://people.netscape.com/hecker/settingupshop.html) is to address the business realities of open-source software. I write as someone who lobbied for an open-source strategy within a major commercial software company (Netscape Communications Corp.) and closely observes and sometimes participates in the implementation of that strategy. Simply put, this article is about how to "set up shop" in the "bazaar."

"Free" Software

Open-source software and open-source development projects have existed for many years under the general term "free software." The word "free" has traditionally led commercial software vendors to think "no revenue," and customers of those companies to think "no support." Thus most in the commercial world saw free software as irrelevant, and free-software developers as idealistic and naive. Similarly, the writings of some free-software advocates have seemed (in tone if not in substance) to portray commercial software companies as interested only in short-term profits at the expense of the long-term interests of users and the software development community as a whole.

Perceptions about free software—pro and con—have often been more absolute than reality warrants. Even Richard Stallman, a free-software advocate often thought hostile to commercial considerations, did not argue in his "GNU Manifesto" (http://www.gnu.org/gnu/manifesto.html) that software development should always be an unpaid or nonprofit activity. Rather he proposed that for-profit business models should treat software as a professional service rather than as intellectual property. Similarly, many companies have used software originating in the free-software community as the basis for commercial products, and in some cases have contributed to the development of free software through donations of money, hardware, or their employees' time.

Events like the growing success of the Linux operating system (developed under the free-software model) and Netscape's release of Communicator source code have focused more public attention on the potential importance of free software to businesses, as both users and for-profit producers of software. These events have also led to the recasting of "free" software as "open-source" (http://www.opensource.org/history.html) software, a term that emphasizes the importance of making source code freely available. It may also remind people (at least subliminally) that a company can choose to make source code freely available and still serve its own business interests as a for-profit organization.

Nonetheless, open-source software is still "free" in that no license fees are charged for use or redistribution of binaries or source code, and users are free to modify the source, create derivative works, and distribute those works. These freedoms are essential to many free-software developers, while others emphasize the ability of open-source development to produce better software. For a good discussion of the tension between these views see Aaron Renn's paper "'Free', 'Open Source', and Philosophies of Software Ownership" (http://www.urbanophile.com/arenn/hacking/fsvos.html).

A Counterintuitive Strategy

Still, this strategy may seem counterintuitive or even self-destructive, as it goes against years of tried and true commercial practice. But consider an his-

FREQUENTLY ASKED QUESTIONS

"How can I make any money if I give the software away?"

Just because you no longer charge traditional software license fees doesn't mean that you can't sell the software in some form. For example, you could sell the software in a traditional retail package with CD and hard-copy manual included, while still allowing users to acquire the software at no charge through other means (for example, downloading it over the Internet). Users who are not price-sensitive may prefer to buy the retail product based on its convenience and other attributes they find valuable. You can also derive revenue from ancillary products and services as discussed elsewhere in this article.

"If I make a product open-source, does that mean I'm no longer committed to it?"

Even after you convert a product to open-source, you can and should continue to have ultimate responsibility for it. Not only will you exercise influence and oversight over the product's evolution, but you can and should continue to release an "official" binary version of the product, packaged for easy installation by users and with full QA testing, product support, and branding.

"But customers don't really want to have source code and can't take advantage of it anyway. Why would they prefer an open-source product over a prepackaged binary product?"

Some customers do indeed want source code; others who prefer not to deal with source code can use the prepackaged "official" product binaries you'll continue to produce. Both sets of customers benefit from the improved product quality and enhancements resulting from open-source development.

"Wouldn't this lead to fragmentation of the product into incompatible versions?"

This is one of the most common objections but need not be an issue, both because of the particular dynamics of open-source development as they have evolved over time and because your company can actively minimize the possibility of this happening. In particular, the free-software developer community has unwritten but historically effective rules that assign control of an open-source project, including the right to designate "official" versions, to a single entity (an individual, an informal group, or a formal organization). As the product's original developer, you are the natural candidate, assuming you do the necessary things to live up to your assigned role.

"What about the risk to customers from 'rogue' versions?"

Again, this has not proved problematic with open-source products, both because of public review and because there is typically a single source (the original vendor) of the "official" version that has undergone additional review and testing.

"What about providing technical support to customers with modified versions?"

Continued on the next page

torical analogy. When Netscape first made the Navigator Web browser available for unrestricted download over the Internet, many questioned how Netscape could possibly make money "giving the software away." In retrospect, many see this strategy as a successful innovation that was key to Netscape's rapid growth. The current interest in open-source may signal another such industry-changing event.

Certainly Netscape's own experience with open-source development has been almost entirely positive. Since the Communicator source code was released, Netscape has received many contributions of code from free-software developers, including simple bug fixes, patches for ports to new platforms, a new system of scripts and makefiles for building the code, and an entire parser for the XML language; Netscape has also had the benefit of detailed code reviews, suggested design changes, and expert advice in various areas, most notably concerning interpretation of W3 Consortium standards for HTML and CSS. In December 1998 Netscape released its first preview of software developed mainly under the open-source model, its "Gecko" technology for rendering Web pages (http://home.netscape.com/newsref/pr/newsrelease711.html).

Other companies besides Netscape have also benefited from the current interest in open-source development; in particular, several companies providing open-source software have received funding from venture capitalists and others. For example, Red Hat Software, a distributor of the open-source Linux operating system, received funding from Netscape, Intel, and two venture capital firms (http://www.redhat.com/release.phtml?id=58). (Among other reasons for this investment, the success of Red Hat

If you wish not to support modified versions of your product, leave this to the general free-software community or to other companies. You can also contract out to external developers to provide such support as part of your own support offerings.

"Wouldn't customers be concerned with my open-source code 'tainting' theirs if they used it for their own projects?"

To allay such concerns you can use a license, such as a BSD-style license, that does not impose license restrictions on derivative works using the open-source code, or one like the MPL that does not impose license restrictions on other code that calls (or is called by) open-source code using a defined API.

"What about embarrassing things that people might discover in our source code?"

You definitely want "bad" things in your source code to be exposed, most notably bugs; open-source development increases the chances that both major and minor bugs will be found and fixed. Other potentially embarrassing things in your source code, such as inappropriate language in comments, can and should be removed when you prepare the source for public release.

"Wouldn't releasing my source code expose confidential plans and strategies to competitors?"

Moving to an open-source model does imply sharing your product strategies with external developers and letting them influence those strategies; it also implies sharing those strategies with competitors. However, this can also result in greater public support for your strategies (because the outside world is helping you create them), helping to counter those of your competitors. Also, releasing source does not imply or require making all internal information publicly available; in particular you can continue to closely control confidential details of business plans and the like.

"Wouldn't people just use our code and our expertise without our getting anything in return?"

This objection resembles the original objections to companies like Netscape allowing downloading of software over the Internet at a time when vendors saw software piracy as a major cause of lost revenue. The answer to the objection is also similar: that the benefits of a properly executed open-source strategy can well outweigh the costs. For example, every bug discovered and fixed by a developer "out there" directly saves you money otherwise required for in-house QA and software maintenance. It also increases the value of your software as a reliable product and can indirectly lead to increased revenues for other products and services associated with that software.

"What about competitors who might try to 'hijack' an open-source product for their own purposes?"

Open-source licenses such as the GPL and MPL can be used to enforce public disclosure and sharing of source code modifications. In the open-source world, competitors must play by the same rules as everyone else, and those rules have evolved to minimize the chances of one individual or organization exercising undue advantage.

Software and other Linux vendors potentially helps vendors such as Netscape that provide software for Linux and hardware vendors such as Intel that sell processors on which Linux will run.)

Using Open Source to Meet Business Challenges

In the traditional software business model, your company provides all (or almost all) of the value to customers, and you realize revenues and profits in return through traditional software license fees. In an open-source business model, much of the value provided to customers will not be provided solely by you, but rather by other developers who are attracted to working on your open-source products and who will thus help augment your resources as opposed to your competitors'. These "outside" developers may be motivated by the prospect of working with software that solves important problems for them and for others, the possibility of future gain providing related services and creating related products, the opportunity to increase their own personal knowledge, or the ego satisfaction of enhancing their reputation among their peers.

Thus, much of your potential success will depend on the efforts of others willing to work for you "for free"—you need free-software developers who will contribute their work to your company, and to the developer community at large, without demanding or receiving money or other tangible payment in return. However, many free-software developers will not (and should not) do such work unless you treat them fairly and provide them with the freedoms and other in-

tangible "payments" that they do want (and demand). This stems in part from your company's attitudes and actions toward developers working with its products, but is also formalized in the company's choice of an open-source license, specifying the terms and conditions under which the company's open-source products can be used, modified, and redistributed.

Open-Source Licensing

Several standard license agreements have been published for use with open-source software; some work better than others for particular business models. All share some features (http://www.opensource.org/osd.html), most notably making software "free" to users both by being no-cost and by minimizing restrictions on use and redistribution. These features are needed for developers to feel fairly treated and satisfied with the intangible rewards of working with your software. Try to use one of these existing licenses, or modify one to meet your needs:

- You can release your software into the public domain, with no license at all.
- Licenses like the BSD (Berkeley Software Distribution) License (http://www.opensource.org/bsd-license.html) place relatively few constraints on what a developer may do (including creating proprietary versions of open-source products).
- The GNU General Public License (GPL) (http://www.gnu.org/copyleft/gpl.html) and variants attempt to constrain developers from making changes to open-source products for commercial purposes and then not contributing those changes back to the developer community.
- The Artistic License (http://language.perl.com/misc/Artistic.html) modifies some of the more controversial aspects of the GPL.
- The Mozilla Public License (MPL) (http://www.mozilla.org/NPL/MPL-1.0.html) and variants (including the Netscape Public License or NPL) go further than the BSD and similar licenses in encouraging the release of derivative works as open-source while still allowing developers to create proprietary add-ons if they wish.

So How Do You Make a Profit?

Since you cannot use traditional software licenses and license fees with open-source software, you must find other ways of generating revenues and profits based on the value you are providing to customers. Doing this successfully requires selecting a suitable business model and executing it well. Several business models are available (names and descriptions for the first four models are courtesy of OpenSource.Org, http://www.opensource.org/).

- Support Sellers: Revenue comes from media distribution, branding, training, consulting, custom development, and post-sales support.
- Loss Leader: A no-charge open-source product is used as a loss leader for traditional commercial software.
- Widget Frosting: Companies in business primarily to sell hardware use the open-source model for enabling software such as driver and interface code.
- Accessorizing: A company distributes books, computer hardware, and other physical items associated with and supportive of open-source software.
- Service Enabler: Open-source software is created and distributed primarily to support access to revenue-generating online services.
- Brand Licensing: One company charges other companies for the right to use its brand names and trademarks in creating derivative products.
- Sell It, Free It: A company's software products start out their product life cycle as traditional commercial products and then are continually converted to open-source products when appropriate.
- Software Franchising: This combines several of the preceding models (in particular Brand Licensing and Support Sellers). A company authorizes others to use its brand names and trademarks in creating associated organizations doing custom software development in particular geographic areas or vertical markets. The company might also supply franchises with training and related services in exchange for franchise fees of some sort.

Making a profit requires selecting a suitable business model and executing it well.

Certain types of hybrid business models relax the constraints surrounding open-source. For example, a company might use both traditional licensing and open-source-like licensing "side by side" for the same product, differentiating between users or between different types of use. Alternately, a company might license source widely to any and all users, and even allow "evaluation" licensing at no charge, but still charge "right-to-modify" license fees and restrict redistribution of modified versions in some way. These

business models are not true open-source models (http://www.opensource.org/osd.html), and many free-software developers view them unfavorably, in some cases starting competitive development efforts simply to provide a more open alternative to software released under a hybrid model. Some companies, for example Troll Tech (http://www.troll.no), have abandoned such models and moved to full open-source licensing; it remains to be seen how successful these types of hybrid business models can be in practice.

Implementing an Open-Source Strategy

Beyond selecting an appropriate license and business model, what else might your company have to do to implement an open-source strategy?

♦ Code sharing. For historical reasons your open-source product may share a common source code base with others of your products that will remain proprietary. If so, make sure that open-source development can proceed without complicating your other internal development efforts; this may require special licensing considerations and appropriate modularization to enforce a clean separation.

♦ Third-party technology. If your product includes technology licensed from third parties, you must treat such third-party code specially to create a releasable open-source product. You might remove the code entirely, seek permission to include third-party code (perhaps under some special arrangement), or replace it with open-source code providing equivalent or similar functionality. The presence of third-party technology also can influence your choice of open-source license.

♦ Code sanitization. To ensure your source code is ready for public distribution, you must remove or revise inappropriate language, comments intended for internal viewing only, and so on.

♦ Export control. If your company is located in the US and your product contains security and cryptographic code, to obtain export approval you will almost certainly have to modify your product, including removing all encryption code and security code that calls that code (http://www.mozilla.org/crypto-faq.html).

♦ Product development processes. Releasing source for a product will almost certainly change the way you do product development. Indeed, without such changes you will not realize most of the benefits of an open-source strategy. Some tips: form a team responsible for your open-source efforts; provide infrastructure for external developers (newsgroups, source code repositories with revision control, special bug-reporting systems); and have your own developers be "customers" of that infrastructure, in principle no different than any other developers.

Open-source development processes

Suppose that you release one of your products (whether existing or new) as open-source. How do you coordinate the efforts of potentially hundreds or even thousands of developers worldwide who might be creating bug fixes, customized versions, and add-on products all based on your source code?

At first glance, this seems impossible. However, at least one successful project exists involving distributed and relatively loosely coordinated development of software products—a project arguably as complex and functional as any commercial product. That example is the project that created the Linux operating system kernel and the body of free software (from the GNU project and elsewhere) that runs on top of it. The Linux experience offers many lessons on how to organize software development following an open-source strategy, and in particular shows the benefits of open-source development in improving the overall quality of Linux and GNU software and enabling new features and add-on software to be easily and quickly added. These benefits have in turn increased the attractiveness of Linux as an alternative operating system platform. Raymond discusses these lessons at length in "The Cathedral and the Bazaar," but following are some of the key points and their implications.

Properly organized and coordinated, distributed development can produce more products faster and with higher quality than would be possible in an isolated effort.

Properly organized and coordinated, distributed development can produce more products faster and with higher quality than would be possible in an isolated effort. As Raymond puts it, "The developer who uses only his or her own brain in a closed project is going to fall behind the developer who knows

how to create an open, evolutionary context in which bug-spotting and improvements get done by hundreds of people." Substitute "software company" for "developer" and you have a basic theme of this article; this approach offers you one way to compete successfully with larger competitors without having their internal resources.

Distributed development is jump-started and proceeds most rapidly when there exists a body of code in which developers can see the promise of solutions to problems that interest them, and which developers can use as a base for their own work to solve those problems. One of your major roles would be to provide this existing source code base initially, and to continue to seed it with new contributions in the form of new product features with accompanying source code.

Higher-quality code can be generated faster when you enlist other people to do not only bug detection and reporting but bug fixing as well. Bug fixers are in turn more motivated when you do frequent releases that incorporate their fixes and enable them to see the fruits of their efforts. Here you would need ways to ensure that other developers' bug fixes are in fact incorporated back into your source tree. This is both easier and harder than using the Internet as your beta test site, as many software companies are now doing by releasing beta versions for general download: On the one hand, developers with access to source would be much more likely to not only find problems but also reproduce, diagnose, and fix them. On the other hand, they would have a much higher expectation of you taking their contributions seriously. They would request or even demand access to the actual developers working on the components, and would be even more turned off than regular users if they received little or no response to their communications with you.

The Internet and the collaborative tools built on top of it (e-mail, newsgroups, and so on) form an indispensable infrastructure for coordinating distributed developers. However, you must provide the proper social context, including having development coordinators with the necessary social and communications skills to "lead without coercion" and focus developers' energies into productive channels. You must also select the teams and team leaders carefully to develop products according to this strategy; in particular find developers who have had previous experience with free-software projects.

Whatever benefits open-source software offers for meeting the challenges of software businesses, there is no "free lunch." You cannot simply release source code, put a few newsgroups up, and expect distributed development to magically self-organize.

However, the potential rewards are worth the effort. Open-source software is a "new" business tool that offers the potential to achieve results that are impossible with traditional software development practices alone. ❖

About the Author

Frank Hecker is lead systems engineer for the government sales group of Netscape Communications Corporation; he has worked in similar sales support positions for Tandem Computers, Visix Software, and Prime Computer. Besides open-source software, his professional interests include information systems security and cryptography.

Hecker has BS degrees in applied mathematics and physics from Centre College of Kentucky. He can be reached at hecker@netscape.com; for other contact information, see http://people.netscape.com/hecker.

This document represents Hecker's personal opinions only, and does not necessarily represent the official positions of Netscape Communications Corporation.