Chapter 6

What Makes Open Source Work?

The Linux community seemed to resemble a great babbling bazaar of differing agendas and approaches . . . out of which a coherent and stable system could seemingly emerge only by a succession of miracles.

—Eric Raymond[1]

The earliest open-source programs were those that we now think of as the most basic. When Linus Torvalds began work on his eponymous kernel, he did it not because he wanted to build an operating system, but, as he explained in the 2001 documentary *Revolution OS,* because he wanted to *use* an operating system:

> The thing about an operating system is that you're never supposed to see it. Nobody uses an operating system; people use programs. The only mission in life of an operating system is to help those programs run.[2]

But when Torvalds couldn't find what he needed elsewhere in the community, he started work on his own. And as people joined Torvalds over time, it wasn't because they simply wanted to use the OS, but because they wanted the OS to help them do something.

And when computer scientists at UC Berkeley started work on the building-block networking software that is now an essential component of the Internet and any computer that connects to it—even our laptops and desktop PCs—they weren't doing it because networking was an end in itself. They were doing it because it supported other applications and uses of the computers. (The folks at Berkeley also invented the e-mail infrastructure we all use today.)

Matt Mullenweg's true passion is jazz, but the expat Texan started doing Web sites to pay for sax lessons.[3] But Mullenweg soon came to appreciate the beauty of a well-designed page and good typography and began to struggle with the limitations of the tools that he had to achieve that beauty.

> At the time, everybody was using "nl2br," a function that converted new lines to breaks, but I wanted it to do better.[4]

The problem was that breaks, the
 tag, made a piece of text look correct, but they weren't semantically correct. That is, a break gave the appearance of paragraphs in the text, but they didn't work like paragraphs. And that meant that some typography rules didn't work. How could you tell the Web browser to make the first few words of the first paragraph of each section bigger if the Web browser didn't know where the paragraphs were?

And fixing that was just a start. Mullenweg wanted to automatically insert curly quotes, the quotes that smartly lean left or right on each side of the quoted text, and a dozen other things that might fix what he thought was the ugliness of so much text on the Web.

So Mullenweg, who admits he hadn't done much programming before that, started work on a new function that did what he wanted. He sought help from friends, people on mailing lists, even his dad, and eventually put together the first version of "autop."

The code has been modified over time, reused in other projects, and generally adopted everywhere to the point that the features it provides have become commonplace and expected in any software that publishes to the Web.

And so a fellow who would have rather been playing in jazz clubs found himself writing bits of code. And a number of people, who each had their own goals, found those bits of code useful. And some of them contributed fixes and improvements back.

That's how open-source communities take shape.

"Good programmers know what to write. Great ones know what to rewrite (and reuse)," explains Eric Raymond in "The Cathedral and the Bazaar," and most successful open-source projects prove the truth of it.[5]

The development of the Apache Web server offers an interesting look at how programmers will reuse code and communities can form to solve a common problem while achieving different goals.

Rob McCool wrote httpd, a Web server program that ran on Unix, in the early 1990s while working at the National Center for Supercomputing Applications (NCSA), University of Illinois, Urbana–Champaign.[6] NCSA httpd was one of the first and most popular Web server applications, but formal development came to a halt after McCool left NCSA in 1994. Soon, a new group of sysadmins and webmasters began developing and sharing patches to solve problems they encountered.[7] Eventually the group released a new version in 1995, calling it "Apache" in phonetic reference to the number of patches that had been incorporated in the release.[8]

Apache quickly became the most popular Web server software worldwide, a spot that it's held for more than a decade.[9]

Part of Apache's success has been its extensibility. Apache inherited NCSA httpd's Common Gateway Interface (CGI) standard, which allowed Apache and other software to work together to serve content to Web browsers. Apache would handle the details of communicating with the Web client, while the other software would generate the content of the page to be displayed and communicate that back to Apache through the Common Gateway Interface.[10]

CGI was already a de facto standard by the time the World Wide Web Consortium recognized it in 1995.[11] Web-based applications started to take shape as programmers took advantage of the CGI to speed their work. By not having to build the components of the software that communicated with all the Web browsers visiting the site, developers could focus their attention on building the components that made their application unique.

Rasmus Lerdorf collected a set of CGI applications he had been using with Apache and released them in 1995 as Personal Home Page Tools, or PHP.[12] Lerdorf not only developed the first version of PHP, but also contributed to Apache.[13] "It was purely a case of needing a tool to solve real-world Web-related problems," Lerdorf explained.[14] In 1997 he was approached by a group of programmers who wanted to write a new parsing engine for the project. Lerdorf accepted, and along with "a few other people who had been sending patches and code," the newly assembled group released PHP 3—now the "PHP: Hypertext Preprocessor"—in 1998.[15]

> This was probably the most crucial moment during the development of PHP. The project would have died at that point if it had remained a one-man effort and it could easily have died if the newly assembled group of strangers couldn't figure out how to work together towards a common goal. We somehow managed to juggle our egos and other personal events and the project grew.[16]

And grow it did. About 20 million Web sites worldwide have PHP installed, and there are a number of PHP-based open-source projects in every imaginable category.[17] The popularity of PHP and similar tools eventually highlighted a performance problem in the CGI standard, and developers soon built Apache modules to solve the problem. Today, mod_php is just one of over 400 such extensions to Apache, revealing the flexibility that has made it the most popular Web server.[18]

Parallel and codependent development, such as can be seen with Apache and PHP, can be seen in most every open-source project today.

The number of explanations for how open source works is on par with the number of theories of economics, government, or social systems. But everybody I spoke with pointed to one or more of the following essential characteristics of successful open-source projects: critical mass, evolvability, and passion.

Critical Mass

The first release of Linux in September 1991 was initially downloaded by ten people. Five sent back bug fixes. By 1993, there were an estimated 20,000 Linux users worldwide, with about 100 contributing to the code.[19]

Eric Raymond points to the "massively-parallel peer review" as one of the key components to successful open-source projects.[20] And Linus Torvalds is credited with the maxim explaining, "Many eyes make all bugs shallow."[21]

And those eyes include not just programmers, but an entire community of varying interests, skill levels, and backgrounds. Developer and author Forrest Cavalier identified the following three types of participants in open-source communities.[22]

- **The need-driven consumer:** The largest part of any active open-source community is users who participate because the software solves a problem or fulfills a need. Users may report bugs, but they may not be be programmers and they may not be able to or interested in fixing them.
- **The user-developer:** User-developers—Raymond describes them as co-developers—may contribute code or documentation, as well as participate in discussion and advocacy. Their motivation, according to Cavalier, "may be to have fun, learn, make a contribution, or even get something that fulfills a need of use."[23]
- **The core developer:** A small corps of participants will be actively developing and advancing a project. The Linux kernel is managed by a group of six core devel-

opers (Torvalds + 5); Mullenweg credits a team of fewer than ten people with leading WordPress development. Core developers may change over time, but they shoulder the bulk of the work of advancing the project and fixing bugs identified elsewhere in the community.

Licensing a project under the GPL will make it open source, but a project needs a community to use and support it for it to be successful. Like financial markets, open-source communities are most efficient when there are large numbers of participants.[24] Cavalier, responding to Raymond's "Bazaar," was particularly interested in the "effective size" of the community or bazaar:

> The "effective size" of a bazaar: The total of the number of participants motivated and able to contribute the results of individual effort (modifications, enhancements) or provide feedback to the bazaar for a specific activity.
>
> "Specific activity" is very important to effective size. Bazaars may have very large effective sizes for some activities and not others. For example, a bazaar with a size of 5000 may only have an effective size of 5 (or even less than 1) for an unpopular activity such as documentation or regression testing. This may be due to lack of motivation or inability to contribute. (Many bazaar efforts are volunteer efforts.)[25]

In offering an academic explanation of the open-source development model, Joseph Feller and Brian Fitzgerald agreed:

> Users are a critical feature, serving as coders, testers, documenters, and also providing prompt notification of new requirements.[26]

Evolvability

Software evolves. Well, software that lasts evolves. SWISH became Swish-e, b2 became WordPress, and Apache continues to be patched and extended to serve new needs.

This evolution is essential to meeting our changing needs, and the GPL promotes evolvability by protecting the right of any participant in the community to solve a problem in a program. Active communities effectively emulate organic evolution in the code they produce, often testing different solutions in parallel and selecting the most fit bits of code for each new release. And if the community can't agree on the most fit solution, communities can split, as happened when a new group of programmers rejected WordPress and began work on the original b2 code with a project called b2evolution.

Still, some software is more amenable to evolution than others. Writer and NYU professor Clay Shirky found that some systems, especially those promised to be the next "industry standard," are too large and unwieldy to evolve. Writing in 1996 on the evolution of the Web and HTML, Shirky noted:

> Evolvable systems—those that proceed not under the sole direction of one centralized design authority but by being adapted and extended in a thousand small ways in a thousand places at once—have three main characteristics that are germane to their eventual victories over strong, centrally designed protocols.
>
> - Only solutions that produce partial results when partially implemented can succeed. The network is littered with ideas that would have worked had everybody adopted them. Evolvable systems begin partially working right away and then grow, rather than needing to be perfected and frozen. Think VMS vs. Unix, cc:Mail vs. RFC-822, Token Ring vs. Ethernet.
> - What is, is wrong. Because evolvable systems have always been adapted to earlier conditions and are always being further adapted to present conditions, they are always behind the times. No evolving protocol is ever perfectly in sync with the challenges it faces.
> - Finally, Orgel's Rule, named for the evolutionary biologist Leslie Orgel—"Evolution is cleverer than you are." As with the list of the Web's obvious deficiencies above, it is easy to point out what is wrong with any evolvable system at any point in its life. No one seeing Lotus Notes and the NCSA server side-by-side in 1994 could doubt that Lotus had the superior technology; ditto ActiveX vs. Java or Marimba vs. HTTP. However, the ability to understand what is missing at any given moment does not mean that one person or a small central group can design a better system in the long haul.
>
> Centrally designed protocols start out strong and improve logarithmically. Evolvable protocols start out weak and improve exponentially. It's dinosaurs vs. mammals, and the mammals win every time. The Web is not the perfect hypertext protocol, just the best one that's also currently practical. Infrastructure built on evolvable protocols will always be partially incomplete, partially wrong and ultimately better designed than its competition.[27]

Software follows many of the same rules. Apache may not be the best Web server for every use, but its flexible architecture and constant evolution continue to attract a large number of developers who would rather live with—and perhaps fix—the limitations than look elsewhere.

Passion

WordPress's Matt Mullenweg hardly hesitates before answering. "You have to be the most passionate user—passionate to the point of obsession," he offers.[28]

Eric Raymond suggests, "Every good work of software starts by scratching a developer's personal itch," adding, "To solve an interesting problem, start by finding a problem that is interesting to you."[29]

To explain the social context of open-source development, Raymond repeats a quote found in Gerald Weinberg's *Psychology of Computer Programming.* The quote is from *Memoirs of a Revolutionist,* the autobiography of Pyotr Alexeyvich Kropotkin, a nineteenth-century Russian anarchist:

> Having been brought up in a serf-owner's family, I entered active life, like all young men of my time, with a great deal of confidence in the necessity of commanding, ordering, scolding, punishing and the like. But when, at an early stage, I had to manage serious enterprises and to deal with [free] men, and when each mistake would lead at once to heavy consequences, I began to appreciate the difference between acting on the principle of command and discipline and acting on the principle of common understanding. The former works admirably in a military parade, but it is worth nothing where real life is concerned, and the aim can be achieved only through the severe effort of many converging wills.[30]

Raymond goes on:

> To operate and compete effectively, hackers who want to lead collaborative projects have to learn how to recruit and energize effective communities of interest in the mode vaguely suggested by Kropotkin's "principle of understanding."[31]

While problems may get solved by people who care enough to solve them, open-source communities build around participants that are passionate about solving a problem. With no means of applying the traditional management techniques or coercion, leaders in the open-source world emerge based on the passion for a project.

Notes

1. Eric S. Raymond, "The Cathedral and the Bazaar," 1998, on the First Monday Web Site, www.firstmonday.org/issues/issue3_3/raymond (accessed Mar. 19, 2007).
2. Linus Tovalds, interviewed in *Revolution OS,* DVD, documentary by J. T. S. Moore (Wonderview Productions, 2001).
3. Matt Mullenweg (WordPress developer), interview by the author, Aug. 6, 2006.
4. Ibid.
5. Raymond, "The Cathedral and the Bazaar."
6. "About the Apache HTTP Server Project," on the Apache HTTP Server Project Web site, http://httpd.apache.org/ABOUT_APACHE.html (accessed Mar. 19, 2007).
7. "Apache HTTP Server," Wikipedia, http://en.wikipedia.org/wiki/Apache_HTTP_Server (accessed Mar. 19, 2007).
8. "Information on the Apache HTTP Server Project," archived on the Wayback Machine Web site, http://web.archive.org/web/19970615081902/http://www.apache.org/info.html (accessed Mar. 19, 2007).
9. "Netcraft Web Server Survey: By Server," July 2006, Netcraft Web site, http://survey.netcraft.com/Reports/0607/byserver (accessed Mar. 19, 2007).
10. "Common Gateway Interface," Wikipedia, http://en.wikipedia.org/wiki/Common_Gateway_Interface (accessed Mar. 19, 2007).
11. "Common Gateway Interface," last updated Oct. 13, 1999, on the World Wide Web Consortium Web site, www.w3.org/CGI (accessed Mar. 19, 2007).
12. "PHP," Wikipedia, http://en.wikipedia.org/wiki/PHP (accessed Mar. 19, 2007).
13. "Rasmus Lerdorf: Biography," O'Reilly Media Web site, www.oreillynet.com/pub/au/85?x-t=book.view (accessed Mar. 19, 2007).
14. Rasmus Lerdorf, "Do You PHP?" Oracle Web site, www.oracle.com/technology/pub/articles/php_experts/rasmus_php.html (accessed Mar. 19, 2007).
15. Ibid.
16. Ibid.
17. "Usage Stats for December 2006," PHP Web site, www.php.net/usage.php (accessed Mar. 19, 2007).
18. "Apache Module Registry," http://modules.apache.org (accessed Mar. 19, 2007).
19. "Linux: The Making of a Global Hack," sidebar to Josh McHugh, "For the Love of Hacking," *Forbes* 162, no. 3 (Aug. 10, 1998), available online at http://members.forbes.com/forbes/1998/0810/6203094s1_print.html (accessed Mar. 19, 2007).
20. Eric S. Raymond, "The Magic Cauldron," June 1999, available online at www.catb.org/~esr/writings/magic-cauldron/magic-cauldron.txt (accessed Mar. 19, 2007).
21. Linus Torvalds, quoted in John G. Drummond, "Open Source Software and Documents: A Literature and Online

Resource Review," April 5, 2000, available online at www.omar.org/opensource/litreview (accessed Mar. 19, 2007).
22. Forrest J. Cavalier, III, "Some Implications of Bazaar Size," Aug. 11, 1998, on the Mib Software Web site, www.mibsoftware.com/bazdev/0003.htm (accessed Mar. 19, 2007).
23. Ibid.
24. Eugene F. Fama, "Random Walks in Stock Market Prices," *Financial Analysts Journal* 21 (Sept./Oct. 1965): 55–59, available online at www.e-m-h.org/Fama1965a.pdf (accessed Mar. 19, 2007).
25. Cavalier, "Some Implications of Bazaar Size."
26. Joseph Feller and Brian Fitzgerald, "A Framework Analysis of the Open Source Development Paradigm," in *Proceedings of the 21st Annual International Conference on Information Systems*, Brisbane, Australia, Dec. 2000, ed. W. Orlikowski et al., available online at www.csis.ul.ie/staff/bf/oss-icis00.pdf (accessed Mar. 19, 2007).
27. Clay Shirky, "In Praise of Evolvable Systems," 1996 (first appeared in the ACM's net_worker), www.shirky.com/writings/evolve.html (accessed Mar. 19, 2007).
28. Mullenweg interview.
29. Raymond, "The Cathedral and the Bazaar."
30. Pyotr Alexeyvich Kropotkin, quoted in Raymond, "The Cathedral and the Bazaar."
31. Raymond, "The Cathedral and the Bazaar."