24 The Open Source Paradigm Shift

Tim O'Reilly

In 1962, Thomas Kuhn published a groundbreaking book entitled *The Structure of Scientific Revolutions*. In it, he argued that the progress of science is not gradual, but (much as we now think of biological evolution) a kind of punctuated equilibrium, with moments of epochal change. When Copernicus explained the movements of the planets by postulating that they moved around the sun rather than the earth, or when Darwin introduced his ideas about the origin of species, they were doing more than just building on past discoveries, or explaining new experimental data. A truly profound scientific breakthrough, Kuhn (1996, 7) notes, "is seldom or never just an increment to what is already known. Its assimilation requires the reconstruction of prior theory and the re-evaluation of prior fact, an intrinsically revolutionary process that is seldom completed by a single man and never overnight."

Kuhn referred to these revolutionary processes in science as "paradigm shifts," a term that has now entered the language to describe any profound change in our frame of reference.

Paradigm shifts occur from time to time in business as well as in science. And as with scientific revolutions, they are often hard fought, and the ideas underlying them not widely accepted until long after they were first introduced. What's more, they often have implications that go far beyond the insights of their creators.

One such paradigm shift occurred with the introduction of the standardized architecture of the IBM personal computer in 1981. In a huge departure from previous industry practice, IBM chose to build its computer from off-the-shelf components, and to open up its design for cloning by other manufacturers. As a result, the IBM personal computer architecture became the standard, over time displacing not only other personal computer designs, but, over the next two decades, also minicomputers and mainframes.

However, the executives at IBM failed to understand the full consequences of their decision. At the time, IBM's market share in computers far exceeded Microsoft's dominance of the desktop operating system market today. Software was a small part of the computer industry, a necessary part of an integrated computer, often bundled rather than sold separately. What independent software companies did exist were clearly satellite to their chosen hardware platform. So when it came time to provide an operating system for the new machine, IBM decided to license it from a small company called Microsoft, giving away the right to resell the software to the small part of the market that IBM did not control. As cloned personal computers were built by thousands of manufacturers large and small, IBM lost its leadership in the new market. Software became the new sun that the industry revolved around; Microsoft, not IBM, became the most important company in the computer industry.

But that's not the only lesson from this story. In the initial competition for leadership of the personal computer market, companies vied to "enhance" the personal computer standard, adding support for new peripherals, faster buses, and other proprietary technical innovations. Their executives, trained in the previous, hardware-dominated computer industry, acted on the lessons of the old paradigm.

The most intransigent, such as Digital's Ken Olson, derided the PC as a toy, and refused to enter the market until too late. But even pioneers like Compaq, whose initial success was driven by the introduction of "luggable" computers, the ancestor of today's laptop, were ultimately misled by old lessons that no longer applied in the new paradigm. It took an outsider, Michael Dell, who began his company selling mail order PCs from a college dorm room, to realize that a standardized PC was a commodity, and that marketplace advantage came not from building a better PC, but from building one that was good enough, lowering the cost of production by embracing standards, and seeking advantage in areas such as marketing, distribution, and logistics. In the end, it was Dell, not IBM or Compaq, that became the largest PC hardware vendor.

Meanwhile, Intel, another company that made a bold bet on the new commodity platform, abandoned its memory chip business and made a commitment to be the more complex brains of the new design. The fact that most of the PCs built today bear an "Intel Inside" logo reminds us that even within commodity architectures, there are opportunities for proprietary advantage.

What does all this have to do with open source software? you might ask.

My premise is that free and open source developers are in much the same position today that IBM was in 1981 when it changed the rules of the com-

puter industry, but failed to understand the consequences of the change, allowing others to reap the benefits. Most existing proprietary software vendors are no better off, playing by the old rules while the new rules are reshaping the industry around them.

I have a simple test that I use in my talks to see whether my audience of computer industry professionals is thinking with the old paradigm or the new. "How many of you use Linux?" I ask. Depending on the venue, 20 to 80 percent of the audience might raise their hands. "How many of you use Google?" Every hand in the room goes up. And the light begins to dawn. Every one of them uses Google's massive complex of 100,000 Linux servers, but they were blinded to the answer by a mindset in which "the software you use" is defined as the software running on the computer in front of you. Most of the "killer apps" of the Internet—applications used by hundreds of millions of people—run on Linux or FreeBSD. But the operating system, as formerly defined, is to these applications only a component of a larger system. Their true platform is the Internet.

It is in studying these next-generation applications that we can begin to understand the true long-term significance of the open source paradigm shift.

If open source pioneers are to benefit from the revolution we've unleashed, we must look through the foreground elements of the free and open source movements, and understand more deeply both the causes and consequences of the revolution.

Artificial intelligence pioneer Ray Kurzweil[1] once said, "I'm an inventor. I became interested in long-term trends because an invention has to make sense in the world in which it is finished, not the world in which it is started."

I find it useful to see open source as an expression of three deep, long-term trends:

- The *commoditization* of software
- Network-enabled *collaboration*
- Software *customizability* (software as a service)

Long-term trends like these "three Cs," rather than the Free Software Manifesto or the Open Source Definition, should be the lens through which we understand the changes that are being unleashed.

Software as Commodity

In his essay "Some Implications of Software Commodification," Dave Stutz writes:

The word *commodity* is used today to represent fodder for industrial processes: things or substances that are found to be valuable as basic building blocks for many different purposes. Because of their very general value, they are typically used in large quantities and in many different ways. Commodities are always sourced by more than one producer, and consumers may substitute one producer's product for another's with impunity. Because commodities are fungible in this way, they are defined by uniform quality standards to which they must conform. These quality standards help to avoid adulteration, and also facilitate quick and easy valuation, which in turn fosters productivity gains. (Stutz 2004b)

Software commoditization has been driven by standards, and in particular by the rise of communications-oriented systems such as the Internet, which depend on shared protocols, and define the interfaces and datatypes shared between cooperating components rather than the internals of those components. Such systems necessarily consist of replaceable parts. A Web server such as Apache or Microsoft's IIS, or browsers such as Internet Explorer, Netscape Navigator, or Mozilla, are all easily swappable, because in order to function, they must implement the HTTP protocol and the HTML data format. Sendmail can be replaced by Exim or Postfix or Microsoft Exchange, because all must support e-mail exchange protocols such as SMTP, POP, and IMAP. Microsoft Outlook can easily be replaced by Eudora, or pine, or Mozilla mail, or a Web mail client such as Yahoo! Mail for the same reason.

(In this regard, it's worth noting that Unix, the system on which Linux is based, also has a communications-centric architecture. In *The Unix Programming Environment*, Kernighan and Pike (1984) eloquently describe how Unix programs should be written as small pieces designed to cooperate in "pipelines," reading and writing ASCII files rather than proprietary data formats. Eric Raymond (2003b) gives a contemporary expression of this theme in his book *The Art of Unix Programming*.)

Note that in a communications-centric environment with standard protocols, both proprietary and open source software become commodities. Microsoft's Internet Explorer Web browser is just as much a commodity as the open source Apache Web server, because both are constrained by the open standards of the Web. (If Microsoft had managed to gain dominant market share at both ends of the protocol pipeline between Web browser and server, it would be another matter![2] This example makes clear one of the important roles that open source does play in "keeping standards honest." This role is being recognized by organizations like the W3C, which are increasingly reluctant to endorse standards that have only proprietary or patent-encumbered implementations.

What's more, even software that starts out proprietary eventually becomes standardized and ultimately commodified. Dave Stutz eloquently describes this process:

> It occurs through a hardening of the external shell presented by the platform over time. As a platform succeeds in the marketplace, its APIs, UI, feature-set, file formats, and customization interfaces ossify and become more and more difficult to change. (They may, in fact, ossify so far as to literally harden into hardware appliances!) The process of ossification makes successful platforms easy targets for cloners, and cloning is what spells the beginning of the end for platform profit margins. (Stutz 2004a)

Consistent with this view, the cloning of Microsoft's Windows and Office franchises has been a major objective of the free and open source communities. In the past, Microsoft has been successful at rebuffing cloning attempts by continually revising APIs and file formats, but the writing is on the wall. Ubiquity drives standardization, and gratuitous innovation in defense of monopoly is rejected by users.

What are some of the implications of software commoditization? One might be tempted to see only the devaluation of something that was once a locus of enormous value. Thus, Red Hat founder Bob Young once remarked, "My goal is to shrink the size of the operating system market." (Red Hat however aimed to own a large part of that smaller market!) Defenders of the status quo, such as Microsoft VP Jim Allchin, have made statements that open source is an intellectual property destroyer, and paint a bleak picture in which a great industry is destroyed, with nothing to take its place.

On the surface, Allchin appears to be right. Linux now generates tens of billions of dollars in server hardware–related revenue, with the software revenues merely a rounding error. Despite Linux's emerging dominance in the server market, Red Hat, the largest Linux distribution company, has annual revenues of only $126 million, versus Microsoft's $32 billion. A huge amount of software value appears to have vaporized.

But is it value or overhead? Open source advocates like to say they're not destroying actual value, but rather squeezing inefficiencies out of the system. When competition drives down prices, efficiency and average wealth levels go up. Firms unable to adapt to the new price levels undergo what the economist E.F. Schumpeter called "creative destruction," but what was "lost" returns manyfold as higher productivity and new opportunities.

Microsoft benefited, along with consumers, from the last round of creative destruction as PC hardware was commoditized. This time around, Microsoft sees the commoditization of operating systems, databases, Web

servers and browsers, and related software as destructive to its core business. But that destruction has created the opportunity for the killer applications of the Internet era. Yahoo!, Google, Amazon, eBay—to mention only a few—are the beneficiaries.

And so I prefer to take the view of Clayton Christensen,[3] the author of *The Innovator's Dilemma* and *The Innovator's Solution*. In a recent article, he articulates "the law of conservation of attractive profits" as follows: "When attractive profits disappear at one stage in the value chain because a product becomes modular and commoditized, the opportunity to earn attractive profits with proprietary products will usually emerge at an adjacent stage" (Christensen 2004, 17).

We see Christensen's thesis clearly at work in the paradigm shifts I'm discussing here. Just as IBM's commoditization of the basic design of the personal computer led to opportunities for attractive profits "up the stack" in software, new fortunes are being made up the stack from the commodity open source software that underlies the internet, in a new class of proprietary applications that I have elsewhere referred to as "infoware."[4]

Sites such as Google, Amazon, and Salesforce.com provide the most serious challenge to the traditional understanding of free and open source software. Here are applications built on top of Linux, but they are fiercely proprietary. What's more, even when using and modifying software distributed under the most restrictive of free software licenses, the GPL, these sites are not constrained by any of its provisions, all of which are conditioned on the old paradigm. The GPL's protections are triggered by the act of software distribution, yet Web-based application vendors never distribute any software: it is simply performed on the Internet's global stage, delivered as a service rather than as a packaged software application.

But even more importantly, even if these sites gave out their source code, users would not easily be able to create a full copy of the running application! The application is a dynamically updated database whose utility comes from its completeness and concurrency, and in many cases, from the network effect of its participating users.[5]

And the opportunities are not merely up the stack. There are huge proprietary opportunities hidden inside the system. Christensen notes: "Attractive profits . . . move elsewhere in the value chain, often to subsystems from which the modular product is assembled. This is because it is improvements in the subsystems, rather than the modular product's architecture, that drives the assembler's ability to move upmarket towards more attractive profit margins. Hence, the subsystems become decommoditized and attractively profitable." (Christensen 2004, 17).

We saw this pattern in the PC market with most PCs now bearing the brand "Intel Inside"; the Internet could just as easily be branded "Cisco Inside." But these "Intel Inside" business opportunities are not always obvious, nor are they necessarily in proprietary hardware or software. The open source BIND (Berkeley Internet Name Daemon) package used to run the Domain Name System (DNS) provides an important demonstration.

The business model for most of the Internet's commodity software turned out to be not selling that software (despite shrinkwrapped offerings from vendors such as NetManage and Spry, now long gone), but services based on that software. Most of those businesses—the Internet Service Providers (ISPs), who essentially resell access to the TCP/IP protocol suite and to e-mail and Web servers—turned out to be low-margin businesses. There was one notable exception.

BIND is probably the single most mission-critical program on the Internet, yet its maintainer has scraped by for the past two decades on donations and consulting fees. But meanwhile, domain name registration—an information service based on the software—became a business generating hundreds of millions of dollars a year, a virtual monopoly for Network Solutions, which was handed the business on government contract before anyone realized just how valuable it would be. The Intel Inside opportunity of the DNS was not a software opportunity at all, but the service of managing the namespace used by the software. By a historical accident, the business model became separated from the software.

That services based on software would be a dominant business model for open source software was recognized in *The Cathedral and the Bazaar*, Eric Raymond's (2001) seminal work on the movement. But in practice, most early open source entrepreneurs focused on services associated with the maintenance and support of the software, rather than true software as a service. (That is to say, software as a service is not service in support of software, but software in support of user-facing services!)

Dell gives us a final lesson for today's software industry. Much as the commoditization of PC hardware drove down IBM's outsize margins but vastly increased the size of the market, creating enormous value for users, and vast opportunities for a new ecosystem of computer manufacturers for whom the lower margins of the PC still made business sense, the commoditization of software will actually expand the software market. And as Christensen (2004, 17) notes, in this type of market, the drivers of success "become speed to market and the ability responsively and conveniently to give customers exactly what they need, when they need it."

Following this logic, I believe that the process of building custom distributions will emerge as one of the key competitive differentiators among Linux vendors. Much as a Dell must be an arbitrageur of the various contract manufacturers vying to produce fungible components at the lowest price, a Linux vendor will need to manage the ever-changing constellation of software suppliers whose asynchronous product releases provide the raw materials for Linux distributions. Companies like Debian founder Ian Murdock's Progeny Systems already see this as the heart of their business, but even old-line Linux vendors like SuSe and new entrants like Sun tout their release engineering expertise as a competitive advantage.[6]

But even the most successful of these Linux distribution vendors will never achieve the revenues or profitability of today's software giants like Microsoft or Oracle, unless they leverage some of the other lessons of history. As demonstrated by both the PC hardware market and the ISP industry (which as noted previously is a service business built on the commodity protocols and applications of the Internet), commodity businesses are low-margin for most of the players. Unless companies find value up the stack or through an "Intel Inside" opportunity, they must compete only through speed and responsiveness, and that's a challenging way to maintain a pricing advantage in a commodity market.

Early observers of the commodity nature of Linux, such as Red Hat's founder Bob Young, believed that advantage was to be found in building a strong brand.[7] That's certainly necessary, but it's not sufficient. It's even possible that contract manufacturers such as Flextronix, which work behind the scenes as industry suppliers rather than branded customer-facing entities, may provide a better analogy than Dell for some Linux vendors.

In conclusion, software itself is no longer the primary locus of value in the computer industry. The commoditization of software drives value to services enabled by that software. New business models are required.

Network-Enabled Collaboration

To understand the nature of competitive advantage in the new paradigm, we should look not to Linux, but to the Internet, which has already shown signs of how the open source story will play out.

The most common version of the history of free software begins with Richard Stallman's ethically motivated 1984 revolt against proprietary software. It is an appealing story centered on a charismatic figure, and leads straight into a narrative in which the license he wrote—the GPL—is the

centerpiece. But like most open source advocates, who tell a broader story about building better software through transparency and code sharing, I prefer to start the history with the style of software development that was normal in the early computer industry and academia. Because software was not seen as the primary source of value, source code was freely shared throughout the early computer industry.

The Unix software tradition provides a good example. Unix was developed at Bell Labs, and was shared freely with university software researchers, who contributed many of the utilities and features we take for granted today. The fact that Unix was provided under a license that later allowed ATT to shut down the party when it decided it wanted to commecialize Unix, leading ultimately to the rise of BSD Unix and Linux as free alternatives, should not blind us to the fact that the early, collaborative development preceded the adoption of an open source licensing model. Open source licensing began as an attempt to preserve a culture of sharing, and only later led to an expanded awareness of the value of that sharing.

For the roots of open source in the Unix community, you can look to the research orientation of many of the original participants. As Bill Joy noted in his keynote at the O'Reilly Open Source Convention in 1999, in science, you share your data so other people can reproduce your results. And at Berkeley, he said, we thought of ourselves as computer scientists.[8]

But perhaps even more important was the fragmented nature of the early Unix hardware market. With hundreds of competing computer architectures, the only way to distribute software was as source! No one had access to all the machines to produce the necessary binaries. (This demonstrates the aptness of another of Christensen's "laws," the law of conservation of modularity. Because PC hardware was standardized and modular, it was possible to concentrate value and uniqueness in software. But because Unix hardware was unique and proprietary, software had to be made more open and modular.)

This software source code exchange culture grew from its research beginnings, but it became the hallmark of a large segment of the software industry because of the rise of computer networking.

Much of the role of open source in the development of the Internet is well known: the most widely used TCP/IP protocol implementation was developed as part of Berkeley networking; BIND runs the DNS, without which none of the Web sites we depend on would be reachable; sendmail is the heart of the Internet e-mail backbone; Apache is the dominant Web server; Perl is the dominant language for creating dynamic sites, and so on.

Less often considered is the role of Usenet in mothering the Net we now know. Much of what drove public adoption of the Internet was in fact Usenet, that vast distributed bulletin board. You "signed up" for Usenet by finding a neighbor willing to give you a newsfeed. This was a true collaborative network, where mail and news were relayed from one cooperating site to another, often taking days to travel from one end of the Net to another. Hub sites formed an ad hoc backbone, but everything was voluntary.

Rick Adams, who created UUnet, the first major commercial ISP, was a free software author (though he never subscribed to any of the free software ideals—it was simply an expedient way to distribute software he wanted to use). He was the author of B News (at the time the dominant Usenet news server) as well as SLIP (Serial Line IP), the first implementation of TCP/IP for dialup lines. But more importantly for the history of the Net, Rick was also the hostmaster of the world's largest Usenet hub. He realized that the voluntary Usenet was becoming unworkable, and that people would pay for reliable, well-connected access. UUnet started out as a nonprofit, and for several years, much more of its business was based on the earlier UUCP (Unix-Unix Copy Protocol) dialup network than on TCP/IP. As the Internet caught on, UUnet and others liked it helped bring the Internet to the masses. But at the end of the day, the commercial Internet industry started out of a need to provide infrastructure for the completely collaborative UUCPnet and Usenet.

The UUCPnet and Usenet were used for e-mail (the first killer app of the Internet), but also for software distribution and collaborative tech support. When Larry Wall (later famous as the author of Perl) introduced the patch program in 1984, the ponderous process of sending around nine track tapes of source code was replaced by the transmission of "patches"—editing scripts that update existing source files. Add in Richard Stallman's GNU C compiler (gcc), and early source code control systems like RCS (eventually replaced by CVS and now Subversion), and you had a situation where anyone could share and update free software. Early Usenet was as much a "Napster" for shared software as it was a place for conversation.

The mechanisms that the early developers used to spread and support their work became the basis for a cultural phenomenon that reached far beyond the tech sector. The heart of that phenomenon was the use of wide-area networking technology to connect people around interests, rather than through geographical location or company affiliation. This was the beginning of a massive cultural shift that we're still seeing today. This cultural shift may have had its first flowering with open source software, but

it is not intrinsically tied to the use of free and open source licenses and philosophies.

In 1999, together with with Brian Behlendorf of the Apache project, O'Reilly founded a company called Collab.Net to commercialize not the Apache product but the Apache process. Unlike many other OSS projects, Apache wasn't founded by a single visionary developer but by a group of users who'd been abandoned by their original "vendor" (NCSA) and who agreed to work together to maintain a tool they depended on. Apache gives us lessons about intentional wide-area collaborative software development that can be applied even by companies that haven't fully embraced open source licensing practices. For example, it is possible to apply open source collaborative principles inside a large company, even without the intention to release the resulting software to the outside world.

While Collab.Net is best known for hosting high-profile corporate-sponsored open source projects like OpenOffice.Org, its largest customer is actually HP's printer division, where Collab's SourceCast platform is used to help more than 3,000 internal developers share their code within the corporate firewall. Other customers use open source–inspired development practices to share code with their customers or business partners, or to manage distributed worldwide development teams.

But an even more compelling story comes from that archetype of proprietary software, Microsoft. Far too few people know the story of the origin of ASP.Net. As told to me by its creators, Mark Anders and Scott Guthrie, the two of them wanted to re-engineer Microsoft's ASP product to make it XML-aware. They were told that doing so would break backward compatibility, and the decision was made to stick with the old architecture. But when Anders and Guthrie had a month between projects, they hacked up their vision anyway, just to see where it would go. Others within Microsoft heard about their work, found it useful, and adopted pieces of it. Some six or nine months later, they had a call from Bill Gates: "I'd like to see your project."

In short, one of Microsoft's flagship products was born as an internal "code fork," the result of two developers "scratching their own itch," and spread within Microsoft in much the same way as open source projects spread on the open Internet. It appears that open source is the "natural language" of a networked community. Given enough developers and a network to connect them, open source–style development behavior emerges.

If you take the position that open source licensing is a means of encouraging Internet-enabled collaboration, and focus on the end rather than the

means, you'll open a much larger tent. You'll see the threads that tie together not just traditional open source projects, but also collaborative "computing grid" projects like SETI@home, user reviews on Amazon.com, technologies like collaborative filtering, new ideas about marketing such as those expressed in the Cluetrain Manifesto, weblogs, and the way that Internet message boards can now move the stock market. What started out as a software development methodology is increasingly becoming a facet of every field, as network-enabled conversations become a principal carrier of new ideas.

I'm particularly struck by how collaboration is central to the success and differentiation of the leading Internet applications. eBay is an obvious example, almost the definition of a "network effects" business, in which competitive advantage is gained from the critical mass of buyers and sellers. New entrants into the auction business have a hard time competing, because there is no reason for either buyers or sellers to go to a second-tier player.

Amazon.com is perhaps even more interesting. Unlike eBay, whose constellation of products is provided by its users, and changes dynamically day to day, products identical to those Amazon sells are available from other vendors. Yet Amazon seems to enjoy an order-of-magnitude advantage over those other vendors. Why? Perhaps it is merely better execution, better pricing, better service, better branding. But one clear differentiator is the superior way that Amazon has leveraged its user community.

In my talks, I give a simple demonstration. I do a search for products in one of my publishing areas, JavaScript. On Amazon, the search produces a complex page with four main areas. On the top is a block showing the three most popular products. Down below is a longer search listing that allows the customer to list products by criteria such as bestselling, highest-rated, price, or alphabetically. On the right and the left are user-generated "Listmania" lists. (These lists allow customers to share their own recommendations for other items related to a chosen subject.)

The section labeled "most popular" might not jump out at first. But as a vendor who sells to Amazon.com, I know that it is the result of a complex, proprietary algorithm that combines not just sales but also the number and quality of user reviews, user recommendations for alternative products, links from Listmania lists, "also bought" associations, and all the other things that Amazon.com refers to as the "flow" around products.

The particular search that I like to demonstrate is usually topped by my own *JavaScript: The Definitive Guide*. As of this writing, the book has 196

reviews, averaging 4½ stars. Those reviews are among the more than ten million user reviews contributed by Amazon.com customers.

Now contrast the #2 player in online books, barnesandnoble.com. The top result is a book published by Barnes & Noble itself, and there is no evidence of user-supplied content. *JavaScript: The Definitive Guide* has only 18 comments, the order-of-magnitude difference in user participation closely mirroring the order-of-magnitude difference in sales.

Amazon.com doesn't have a natural network-effect advantage like eBay, but they've built one by designing their site for user participation. Everything from user reviews, alternative product recommendations, Listmania, and the Associates program (which allows users to earn commissions for recommending books) encourages users to collaborate in enhancing the site. Amazon Web Services, introduced in 2001, take the story even further, allowing users to build alternate interfaces and specialized shopping experiences (as well as other unexpected applications), using Amazon's data and commerce engine as a back end.

Amazon's distance from competitors, and the security it enjoys as a market leader, is driven by the value added by its users. If, as Eric Raymond (2001) said, one of the secrets of open source is "treating your users as co-developers," Amazon has learned this secret. But note that it's completely independent of open source licensing practices! We start to see that what has been presented as a rigidly constrained model for open source may consist of a bundle of competencies, not all of which will always be found together.

Google makes a more subtle case for the network effect story. Google's initial innovation was the PageRank algorithm, which leverages the collective preferences of Web users, expressed by their hyperlinks to sites, to produce better search results. In Google's case, the user participation is extrinsic to the company and its product, and so can be copied by competitors. If this analysis is correct, Google's long-term success will depend on finding additional ways to leverage user-created value as a key part of their offering. Services such as Orkut and Gmail suggest that this lesson is not lost on them.

Now consider a counter-example. MapQuest is another pioneer who created an innovative type of Web application that almost every Internet user relies on. Yet the market is shared fairly evenly between MapQuest (now owned by AOL), maps.yahoo.com, and maps.msn.com (powered by MapPoint). All three provide a commodity business powered by standardized software and databases. None of them have made a concerted effort to leverage user-supplied content, or engage their users in building out the

application. (Note also that all three are enabling an "Intel Inside"–style opportunity for data suppliers such as Navteq, now planning a multibillion-dollar IPO!)

The Architecture of Participation

I've come to use the term "the architecture of participation" to describe the nature of systems that are designed for user contribution. Larry Lessig's (2000) book *Code and Other Laws of Cyberspace*, which he characterizes as an extended meditation on Mitch Kapor's maxim that "architecture is politics," made the case that we need to pay attention to the architecture of systems if we want to understand their effects.

I immediately thought of Kernighan and Pike's (1984) description of the Unix software tools philosophy. I also recalled an unpublished portion of the interview we did with Linus Torvalds to create his essay for the book *Open Sources* (DiBona et al. 1999). Linus too expressed a sense that architecture may be important than source code. "I couldn't do what I did with Linux for Windows, even if I had the source code. The architecture just wouldn't support it." Too much of the Windows source code consists of interdependent, tightly coupled layers for a single developer to drop in a replacement module.

And of course the Internet and the World Wide Web have this participatory architecture in spades. As outlined earlier in the section on software commoditization, any system designed around communications protocols is intrinsically designed for participation. Anyone can create a participating, first-class component.

In addition, the IETF, the Internet standards body has a great many similarities with an open source software project. The only substantial difference is that the IETF's output is a standards document rather than a code module. Especially in the early years, anyone could participate, simply by joining a mailing list and having something to say, or by showing up to one of the three annual face-to-face meetings. Standards were decided on by participating individuals, irrespective of their company affiliations. The very name for proposed Internet standards, RFCs (Request for Comments) reflects the participatory design of the Net. Though commercial participation was welcomed and encouraged, companies (like individuals) were expected to compete on the basis of their ideas and implementations, not their money or disproportional representation. The IETF approach is where open source and open standards meet.

And while there are successful open source projects like Sendmail that are largely the creation of a single individual, and have a monolithic archi-

tecture, those that have built large development communities have done so because they have a modular architecture that allows easy participation by independent or loosely coordinated developers. The use of Perl, for example, exploded along with CPAN, the Comprehensive Perl Archive Network, and Perl's module system, which allowed anyone to enhance the language with specialized functions and make them available to other users.

The Web, however, took the idea of participation to a new level, because it opened that participation not just to software developers but to all users of the system.

It has always baffled and disappointed me that the open source community has not embraced the Web as one of its greatest success stories. Tim Berners-Lee's original Web implementation was not just open source, it was public domain. NCSA's Web server and Mosaic browser were not technically open source, but their source was freely available. While the move of the NCSA team to Netscape sought to take key parts of the Web infrastructure to the proprietary side, and the Microsoft-Netscape battles made it appear that the Web was primarily a proprietary software battleground, we should know better. Apache, the phoenix that grew from the NCSA server, kept the open vision alive, keeping the standards honest, and not succumbing to proprietary embrace and extend strategies.

But even more significantly, HTML, the language of Web pages, opened participation to ordinary users, not just software developers. The "View source" menu item migrated from Tim Berners-Lee's original browser, to Mosaic, and then on to Netscape Navigator and even Microsoft's Internet Explorer. Though no one thinks of HTML as an open source technology, its openness was absolutely key to the explosive spread of the Web. Barriers to entry for "amateurs" were low, because anyone could look "over the shoulder" of anyone else producing a Web page. Dynamic content created with interpreted languages continued the trend toward transparency.

And more germane to my argument here, the fundamental architecture of hyperlinking ensures that the value of the Web is created by its users. In this context, it's worth noting an observation originally made by Clay Shirky (2001) in a talk at my 2001 P2P and Web Services Conference (now renamed the Emerging Technology Conference), entitled "Listening to Napster." There are three ways to build a large database, said Clay. The first, demonstrated by Yahoo!, is to pay people to do it. The second, inspired by lessons from the open source community, is to get volunteers to perform the same task. The Open Directory Project, an open source Yahoo! competitor is the result. (Wikipedia provides another example.) But Napster

demonstrates a third way. Because Napster set its defaults to automatically share any music that was downloaded, every user automatically helped to build the value of the shared database.

This architectural insight may actually be more central to the success of open source than the more frequently cited appeal to volunteerism. The architecture of Linux, the Internet, and the World Wide Web are such that users pursuing their own "selfish" interests build collective value as an automatic byproduct. In other words, these technologies demonstrate some of the same network effect as eBay and Napster, simply through the way that they have been designed.

These projects can be seen to have a natural architecture of participation. But as Amazon demonstrates, by consistent effort (as well as economic incentives such as the Associates program), it is possible to overlay such an architecture on a system that would not normally seem to possess it.

Customizability and Software as Service

The last of my three Cs, customizability, is an essential concomitant of software as a service. It's especially important to highlight this aspect because it illustrates just why dynamically typed languages like Perl, Python, and PHP, so often denigrated by old-paradigm software developers as mere "scripting languages," are so important on today's software scene.

As I wrote in my essay "Hardware, Software and Infoware" (O'Reilly 1997, 192–193):

> If you look at a large web site like Yahoo!, you'll see that behind the scenes, an army of administrators and programmers are continually rebuilding the product. Dynamic content isn't just automatically generated, it is also often hand-tailored, typically using an array of quick and dirty scripting tools.
>
> "We don't create content at Yahoo! We aggregate it," says Jeffrey Friedl, author of the book *Mastering Regular Expressions* and a full-time Perl programmer at Yahoo. "We have feeds from thousands of sources, each with its own format. We do massive amounts of 'feed processing' to clean this stuff up or to find out where to put it on Yahoo!." For example, to link appropriate news stories to tickers at finance.yahoo.com, Friedl needed to write a "name recognition" program able to search for more than 15,000 company names. Perl's ability to analyze free-form text with powerful regular expressions was what made that possible.

Perl has been referred to as the "duct tape of the Internet," and like duct tape, dynamic languages like Perl are important to Web sites like Yahoo!

and Amazon for the same reason that duct tape is important not just to heating system repairmen but to anyone who wants to hold together a rapidly changing installation. Go to any lecture or stage play, and you'll see microphone cords and other wiring held down by duct tape.

We're used to thinking of software as an artifact rather than a process. And to be sure, even in the new paradigm, there are software artifacts, programs, and commodity components that must be engineered to exacting specifications because they will be used again and again. But it is in the area of software that is not commoditized, the glue that ties together components, the scripts for managing data and machines, and all the areas that need frequent change or rapid prototyping, that dynamic languages shine.

Sites like Google, Amazon, or eBay—especially those reflecting the dynamic of user participation—are not just products, they are processes. I like to tell people the story of the Mechanical Turk, a 1770 hoax that pretended to be a mechanical chess-playing machine. The secret, of course, was that a man was hidden inside. The Turk actually played a small role in the history of computing. When Charles Babbage played against the Turk in 1820 (and lost), he saw through the hoax, but was moved to wonder whether a true computing machine would be possible.

Now, in an ironic circle, applications once more have people hidden inside them. Take a copy of Microsoft Word and a compatible computer, and it will still run ten years from now. But without the constant crawls to keep the search engine fresh, the constant product updates at an Amazon or eBay, the administrators who keep it all running, the editors and designers who integrate vendor- and user-supplied content into the interface, and in the case of some sites, even the warehouse staff who deliver the products, the Internet-era application no longer performs its function.

This is truly not the software business as it was even a decade ago. Of course, there have always been enterprise software businesses with this characteristic. (American Airlines' Sabre reservations system is an obvious example.) But only now have they become the dominant paradigm for new computer-related businesses.

The first generation of any new technology is typically seen as an extension to the previous generations. And so, through the 1990's, most people experienced the Internet as an extension or add-on to the personal computer. E-mail and Web browsing were powerful add-ons, to be sure, and they gave added impetus to a personal computer industry that was running out of steam.

(Open source advocates can take ironic note of the fact that many of the most important features of Microsoft's new operating system releases since Windows 95 have been designed to emulate Internet functionality originally created by open source developers.)

But now, we're starting to see the shape of a very different future. Napster brought us peer-to-peer file sharing, SETI@home introduced millions of people to the idea of distributed computation, and now Web services are starting to make even huge database-backed sites like Amazon or Google appear to act like components of an even larger system. Vendors such as IBM and HP bandy about terms like "computing on demand" and "pervasive computing."

The boundaries between cell phones, wirelessly connected laptops, and even consumer devices like the iPod or TiVO, are all blurring. Each now gets a large part of its value from software that resides elsewhere. Dave Stutz (2003) characterizes this as "software above the level of a single device" (http://www.synthesist.net/writing/onleavingms.html).[9]

Building the Internet Operating System

I like to say that we're entering the stage where we are going to treat the Internet as if it were a single virtual computer. To do that, we'll need to create an Internet operating system.

The large question before us is this: what kind of operating system is it going to be? The lesson of Microsoft is that if you leverage insight into a new paradigm, you will find the secret that will give you control over the industry, the "one ring to rule them all," so to speak. Contender after contender has set out to dethrone Microsoft and take that ring from them, only to fail. But the lesson of open source and the Internet is that we can build an operating system that is designed from the ground up as "small pieces loosely joined," with an architecture that makes it easy for anyone to participate in building the value of the system.

The values of the free and open source community are an important part of its paradigm. Just as the Copernican revolution was part of a broader social revolution that turned society away from hierarchy and received knowledge, and instead sparked a spirit of inquiry and knowledge sharing, open source is part of a communications revolution designed to maximize the free sharing of ideas expressed in code.

But free software advocates go too far when they eschew any limits on sharing and define the movement by adherence to a restrictive set of software licensing practices. The open source movement has made a concerted

effort to be more inclusive. Eric Raymond[10] describes the Open Source Definition as a "provocation to thought," a "social contract . . . and an invitation to join the network of those who adhere to it." But even though the open source movement is much more business-friendly and supports the right of developers to choose nonfree licenses, it still uses the presence of software licenses that enforce sharing as its litmus test.

But the lessons of previous paradigm shifts show us a more subtle and powerful story than one that merely pits a gift culture against a monetary culture, and a community of sharers versus those who choose not to participate. Instead, we see a dynamic migration of value, in which things that were once kept for private advantage are now shared freely, and things that were once thought incidental become the locus of enormous value. It's easy for free and open source advocates to see this dynamic as a fall from grace, a hoarding of value that should be shared with all. But a historical view tells us that the commoditization of older technologies and the crystallization of value in new technologies is part of a process that advances the industry and creates more value for all. What is essential is to find a balance, in which we as an industry create more value than we capture as individual participants, enriching the commons that allows for further development by others.

I cannot say where things are going to end. But as Alan Kay[11] once said, "The best way to predict the future is to invent it." Where we go next is up to all of us.

Conclusion

The Open Source Definition and works such as *The Cathedral and the Bazaar* (Raymond 2001) tried to codify the fundamental principles of open source. But as Kuhn notes, speaking of scientific pioneers who opened new fields of study, "Their achievement was sufficiently unprecedented to attract an enduring group of adherents away from competing modes of scientific activity. Simultaneously, it was sufficiently open-ended to leave all sorts of problems for the redefined group of practitioners to resolve. Achievements that share these two characteristics, I shall refer to as 'paradigms'" (Kuhn 1996, 10).

In short, if it is sufficiently robust an innovation to qualify as a new paradigm, the open source story is far from over, and its lessons far from completely understood. Rather than thinking of open source only as a set of software licenses and associated software development practices, we do better to think of it as a field of scientific and economic inquiry, one with

many historical precedents, and part of a broader social and economic story. We must understand the impact of such factors as standards and their effect on commoditization, system architecture and network effects, and the development practices associated with software as a service. We must study these factors when they appear in proprietary software as well as when they appear in traditional open source projects. We must understand how the means by which software is deployed changes the way in which it is created and used. We must also see how the same principles that led to early source code sharing may affect other fields of collaborative activity. Only when we stop measuring open source by what activities are excluded from the definition and begin to study its fellow travelers on the road to the future will we understand its true impact and be fully prepared to embrace the new paradigm.

Notes

1. Speech at the Foresight Senior Associates Gathering, April 2002.

2. See http://salon.com/tech/feature/1999/11/16/microsoft_servers/print.html for my discussion of that subject.

3. I have been talking and writing about the paradigm shift for years, but until I heard Christensen speak at the Open Source Business Conference in March 2004, I hadn't heard his eloquent generalization of the economic principles at work in what I'd been calling business paradigm shifts. I am indebted to Christensen and to Dave Stutz, whose recent writings on software commoditization have enriched my own views on the subject.

4. http://www.oreilly.com/catalog/opensources/book/tim.html.

5. To be sure, there would be many benefits to users were some of Google's algorithms public rather than secret, or Amazon's One-Click available to all, but the point remains: an instance of all of Google's source code would not give you Google, unless you were also able to build the capability to crawl and mirror the entire Web in the same way that Google does.

6. Private communications, SuSe CTO Juergen Geck and Sun CTO Greg Papadopoulos.

7. http://www.oreilly.com/catalog/opensources/book/young.html.

8. I like to say that software enables speech between humans and computers. It is also the best way to talk about certain aspects of computer science, just as equations are the best ways to talk about problems in physics. If you follow this line of reasoning, you realize that many of the arguments for free speech apply to open source

as well. How else do you tell someone how to talk with their computer other than by sharing the code you used to do so? The benefits of open source are analogous to the benefits brought by the free flow of ideas through other forms of information dissemination.

9. Dave Stutz notes (in a private e-mail, 4/29/04, in response to an early draft of this chapter), this software "includes not only what I call 'collective software' that is aware of groups and individuals, but also software that is customized to its location on the network, and also software that is customized to a device or a virtualized hosting environment. These additional types of customization lead away from shrinkwrap software that runs on a single PC or PDA/smartphone and towards personalized software that runs 'on the network' and is delivered via many devices simultaneously."

10. Private e-mail, 4/28/04, in a response to an earlier draft of this chapter.

11. Spoken in a 1971 internal Xerox planning meeting, as quoted in http://www.lisarein.com/alankay/tour.html.