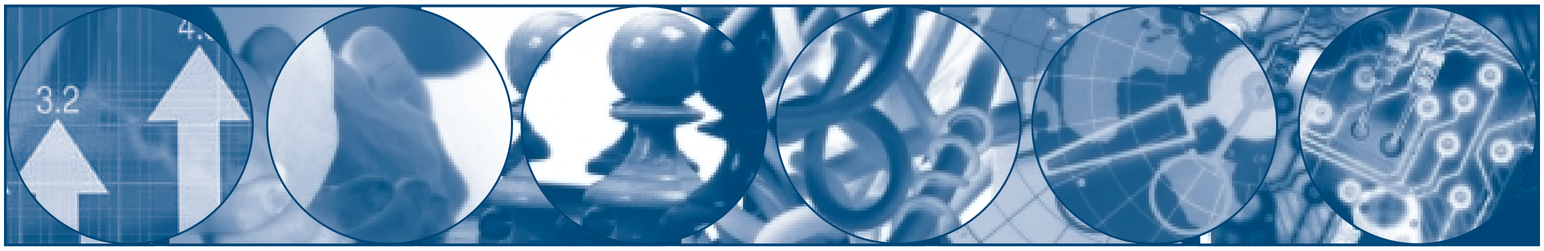
CUTTER CONSORTIUM

Open Source

Moving into the Enterprise

by Gerald S. Greenberg, Ganesh Prasad,
Marc R. Erickson, Luke Hohmann, Michael Olson,
William A. Zucker, and Brian J. Dooley
with an introduction by Jason R. Matthews

CUTTER CONSORTIUM



Cutter Business Technology Council

About Cutter Consortium

Cutter Consortium's mission is to foster the debate of, and dialogue on, the business-technology issues challenging enterprises today and to help organizations leverage IT for competitive advantage and business success. Cutter's philosophy is that most of the issues managers face are complex enough to merit examination that goes beyond simple pronouncements. The Consortium takes a unique view of the business-technology landscape, looking beyond the one-dimensional "technology" fix approach so common today. We know there are no "silver bullets" in IT and that successful implementation and deployment of a technology is as crucial as the selection of that technology.

To accomplish our mission, we have assembled the world's preeminent IT consultants — a distinguished group of internationally recognized experts committed to delivering top-level, critical, objective advice. Each of the Consortium's nine practice areas features a team of Senior Consultants whose credentials are unmatched by any other service provider. This group of experts provides all the consulting, performs all the research and writing, develops and presents all the workshops, and fields all the inquiries from Cutter clients.

This is what differentiates Cutter from other analyst and consulting firms and why we say Cutter gives you access to the experts. All of Cutter's products and services are provided by today's top thinkers in business and IT. Cutter's clients tap into this brain trust and are the beneficiaries of the dialogue and debate our experts engage in at the annual Cutter *Summit*, in the pages of the *Cutter IT Journal*, through the collaborative forecasting of the Cutter Business Technology Council, and in our many reports and advisories.

Cutter Consortium's menu of products and services can be customized to fit your organization's budget. Most importantly, Cutter offers objectivity. Unlike so many information providers, the Consortium has no special ties to vendors and can therefore be completely forthright and critical. That's why more than 5,300 global organizations rely on Cutter for the no-holds-barred advice they need to gain and to maintain a competitive edge — and for the peace of mind that comes with knowing they are relying on the best minds in the business for their information, insight, and guidance.

For more information, contact Cutter Consortium at +1 781 648 8700 or sales@cutter.com.

Open Source

Moving into the Enterprise

CUTTER CONSORTIUM
37 Broadway, Suite 1
Arlington, MA 02474-5552, USA
Telephone: +1 781 648 8700
Fax: +1 781 648 1950
E-mail: sales@cutter.com
Web site: www.cutter.com
ISBN: 1-57484-182-3

Contents

Introduction

by Jason R. Matthews, Senior Consultant, Cutter Consortium

Over the past couple of decades, IT's strategic value in realizing key business goals has been undeniably proven. Add to this the growing use of the Internet as a key service and marketing medium, and businesses from all sectors are finding that IT can help them distinguish themselves from the competition. However, one of the more perplexing problems IT managers face today is how to bring spiraling software costs under control while increasing the use of software to fend off competitive challenges and capitalize on new market opportunities.

IT managers are between the proverbial "rock and a hard place" in view of the fact that profitability, return on investment, and other forms of financial performance top the agendas of corporate executives. Technology investments in B2E portals, application servers, Web services, and disaster recovery have pushed IT budgets slightly up from last year. At the same time, businesses are under more and more pressure from stakeholders to return to profitability, and financial managers have identified IT as the most likely candidate for additional cost cutting. How are CIOs tackling this apparent inconsistency?

One approach many CIOs are seriously investigating is the inclusion of open source technologies as a key part of their overall IT strategy. Open source came out of the Free Software Foundation led by Richard Stallman, who was frustrated by the proprietary nature of traditional vendor software. Open source software is created as projects by communities of software developers who contribute their time, talent, and expertise to create, refine, and evolve some very sophisticated software solutions.

Viewed in the past as the province of "hackers," open source software has become dramatically more popular in corporate circles recently, due in large part to the widely publicized successes of the Linux operating system. Linux exploded onto the operating system stage in the 1990s and now boasts 25% of the network operating system market. The press regularly points to organizations that save millions of dollars annually by replacing their proprietary Windows or Unix operating systems with Linux. And the range of open source software solutions is much broader than just Linux. In fact, the largest open source development Web site, SourceForge.net (www. sourceforge.net), boasts more than 59,000 active software projects.

Due in large part to the contribution model of development, the licensing cost associated with open source software is a mere fraction of the cost of its proprietary counterparts. This is one way Linux and other open source software contribute annually to the savings of millions of IT dollars. License to license, open source software is simply cheaper than traditional proprietary software that's sold worldwide by legions of software salespeople.

This doesn't mean, however, that open source software is "free" in the sense that no money ever changes hands. Companies like Red Hat, JBoss Group, and MySQL AB make a very good living selling their open source software solutions. Instead, open source software is distributed with its underlying source code, which the user is "free to read, modify, and redistribute."
One of the core concepts of open source is the notion that freely available source code encourages more rapid evolution of the software with fewer programming errors. (More eyes mean fewer bugs.) Another core concept is that changes to the underlying source code are then contributed back to the original code base, thereby increasing the software's capabilities.

As strange as this may sound to those who have yet to participate in or benefit from this type of community, the following representative successes of open source software speak volumes for its commercial potential:

- Apache, an open source Web server project, serves up more than 60% of the Web pages on the Internet.
- Sendmail, an open source e-mail transport agent, delivers more than 90% of the e-mail sent over the Internet.
- BIND, an open source domain name service, provides DNS support for virtually the entire Internet.
- JBoss, an open source application server, beat out BEA and IBM for a recent "Best App-Server" editor's choice award; it is downloaded more than 200,000 times a month.
- MySQL, an open source database, boasts more than 4,000,000 installations and counts among its customers such mainstream organizations as Motorola, Sony Pictures, NASA, and HP.

Open source and the principles it promotes are clearly here to stay, and they will flourish in the future. So the most obvious question is, "Why doesn't everyone use it?" It's an interesting question. In fact, it was the crux of our call for papers on this topic. We wanted to know what is right with open source, what is wrong with it, where is it going, and, most importantly, who is using it.

In this report, we start out by looking at the flagship of the open source market — Linux. It seems reasonable to assume that where Linux goes, so too goes the vast majority of the open source marketplace. Jerry Greenberg gives us his viewpoints on the origins of Linux, some interesting success stories of end-user companies benefiting from it, and some guidance on how an organization might adopt Linux. Greenberg heads up the independent Open Source Development Lab (OSDL) and has been active in the computer industry for over four decades. That's more than enough experience for us all to learn from!

In Chapter 2, Ganesh Prasad provides an excellent overview of the enterprise Java offerings available through open source. Nestled into the handy tables he's created are suggestions of open source alternatives to the traditional proprietary (or closed source) infrastructure tools on the market. Looking at some of the offerings, the savvy IT executive can begin to see savings in the tens or hundreds of thousands of dollars … once some hurdles have been overcome.

One of the more complicated hurdles to overcome is how to weave the multicolored threads of open source into a cohesive tapestry suitable for enterprise application development. In Chapter 3, Marc Erickson offers an in-depth view of IBM's recent contribution to the open source movement — Eclipse. Eclipse is an open source software development project that is working to provide a high-quality industry platform for the development of highly integrated tools. According to Erickson, the aptly named Eclipse is changing the very nature of how development environments enable complex application development and deployment.

Of course a development environment, by its very nature, is an integration of many individual technologies to provide a comprehensive solution. In the traditional commercial marketplace, the most complicated aspect of integration is the technical one. But in the open source market, the Achilles' heel has often been the potentially complex licensing terms. According to some licenses, the software you create using open source technologies will require you to release your own software into the open source market. Zounds!

The good news is that Luke Hohmann and Michael Olson share some insights into how best to use and leverage open source licenses. Chapter 4, written as a series of conversations between Hohmann and one of his IT executive clients, tackles each of the predominant licensing issues. Hohmann and Olson show that there are ways for-profit software companies *can* use open source and still make money.

The only area of licensing Hohmann and Olson don't cover (because we asked them not to) is the area of intellectual property. Microsoft and other open source naysayers have pointed to intellectual property, or, rather, the lack of control over it, as a key problem with the whole concept. Stepping into that breach in our open source roundup is William Zucker, an attorney who specializes in technology issues. In Chapter 5, Zucker goes into exquisitely informative detail about such issues as copyright and "copyleft," various types of licenses, and some current legal issues pertaining to open source.

In the final chapter, author Brian J. Dooley examines the open source phenomenon and its place in the enterprise environment. He explores the following questions: "Is open source ready for the enterprise? Can open source and commercial software coexist? What are the benefits of using OSS? What is available for the enterprise? How do the overall costs of OSS compare with those of commercial solutions?" Catch up on this continuing debate to find out how your organization can best take advantage of the latest models in software development, distribution, and pricing.

The open source software movement is clearly in the midst of changing the entire landscape of the commercial software market. The relatively small seismic event of Richard Stallman's refusal to submit to the intellectual restrictions imposed by proprietary software licenses has become a geological shift in the tectonic plates of the commercial software market. Endorsement, active support, and significant technical contributions have come from such computer industry stalwarts as IBM, HP, Sun Microsystems, and others. Traditional proprietary software vendors are constantly developing and repositioning their strategies to compete with or collaborate with the open source movement. (Microsoft, meanwhile, has elected to pursue a debunking approach.)

So where are the corporate CIOs in this adoption process? Well, for those familiar with the typical bell curve of technology adoption, it appears the early adopters are climbing onboard. Where is your organization in its assessment of open source, and how it can bring value to your business? Taken independently or as a whole, the information offered up by the industry experts in this report is bound to make you consider open source as part of your overall IT strategy.

CHAPTER 1

Penguins Stampeding the Enterprise

by Gerald S. Greenberg

When Linus Torvalds first envisioned the type of penguin that would ultimately become "Tux," the definitive icon for Linux, he described a supremely content penguin stuffed with herring. The formalwear motif was just incidental to the species. Today, more than a decade after Torvalds unleashed the first version of the Linux kernel on the world, the black tie look is more appropriate than ever. Linux is steadily growing from its enlightened activist roots and taking up residence deep inside enterprise IT departments.

As little as three years ago, an IT manager who suggested that a Global 2000 company switch from Unix or Windows NT to Linux would have faced a wave of fear, uncertainty, and doubt. Today, however, we can confidently say that Linux is ready for many mission-critical applications requiring five nines of reliability. In the next two years, we at the Open Source Development Lab (OSDL) even expect to see Linux migrate into carrier-grade switches requiring six nines of reliability. Indeed, an IT manager who does not consider Linux over more costly, proprietary systems is putting his or her job in jeopardy.

My own jobs span almost 40 years in enterprise computing, from managing software support for the New York and American Stock Exchanges to running engineering and marketing groups at mainframe computing and Unix companies. Today I run OSDL, a nonprofit organization funded by a global consortium of IT vendors — including Computer Associates, HP, IBM, Intel, and 20 others — with the mission to promote Linux in the enterprise. Our $8-million research lab helps open source developers enhance Linux to meet data center and carrier-grade computing requirements.

Linux is already in the enterprise. There is nothing revolutionary about it. Some of the world's largest companies rely on Linux. I'll describe a few such cases and share some insights we at OSDL have gained about prudent steps to take in adopting Linux. And Linux won't stop at the data center. We predict it will also become the default platform of choice in telecommunications, the most demanding market for reliability and uptime.

So where is Linux today? It is most popular as an operating system for Web, print, and file servers. It is fast gaining popularity in cost-sensitive point-of-sale (POS) terminals and branch office automation. At the opposite — high — end of the scale, Linux has made remarkable inroads into mainframe usage. For example, Merrill Lynch is moving the job of printing its quarterly 401(k) statements to a mainframe running Linux. In the middle — midrange servers — adoption has been somewhat slower, but it is gaining steam as Linux is used to consolidate multiple midrange servers onto one easier-to-manage mainframe. The current downturn in tech spending and shrinking IT budgets have further encouraged CIOs to reduce licensing fees and overall operating expenses by opting for open source solutions such as Linux.

Let Them Eat Herring

Given this progress, it shouldn't be too surprising that Linux adoption in the enterprise is moving quickly up the handle portion of the hockey stick. True, some vendors want to break that handle off, but the very nature of Linux, with its community-owned source code, makes it difficult to undermine by traditional marketing methods. For the moment, six factors are driving Linux adoption:

1. A Linux cluster provides supercomputer performance at a fraction of the expense of proprietary RISC boxes running Unix.
2. Because it's open source, Linux dramatically reduces licensing fees.
3. Linux runs on scores of distinct platforms, including various embedded microprocessors, 32-bit x86 boxes, 64-bit x86 platforms, RISC systems, and the S390/zServer mainframes from IBM. There is a Linux available for every modern general-purpose computer, as well as many special-purpose systems.
4. Linux decouples hardware and operating systems, which in turn reduces traditional large vendor leverage.
5. Linux levels the competitive playing field and promotes vendor neutrality.
6. Linux helps companies attract and retain high-quality technical staff.

In this chapter, I'll take a look at some of the benefits enjoyed by companies that have switched to Linux, and I will also lay out some of the steps to consider if your organization is pondering the penguin.

Branch Office Automation

Even in boom times, traditional retailers often run on slim margins. As companies squeeze every last dime out of the margins, running licensing fee–free Linux on vanilla boxes can be a no-brainer. However, in the long run, the cost savings of not having to do one vendor-forced upgrade will be much greater than even the license savings. For this reason, point-of-sale is a rapidly growing application area for Linux.

The Knoxville, Tennessee, USA–based Regal Entertainment used to ring up concession sales on electronic cash registers at 520 theaters in 36 US states that were not tied to the back end at headquarters. Now Regal is using 2,400 IBM SurePOS 500 systems linked to an IBM eServer iSeries server at its headquarters. The results have been so satisfactory that Regal is now testing in-theater kiosks running on Linux to sell tickets. Regal was also encouraged by the ease of integrating the new Linux system with Java-based programs already in place.

Anchovies Meet Herring

In another large-scale POS application, Papa John's International is moving nearly $1 billion worth of pizza annually and is rolling out a Linux-based POS system to its 2,900 restaurants. Terry Foster, director of field systems development at Papa John's, expects the new system to recover faster from crashes and to save the company money by allowing remote upgrades to the OS. Papa John's is replacing its old Unix-based system and figures major savings will be realized as the company stretches the lifespan of existing hardware with the changeover. Add-ons already designed around the Linux kernel will allow the system to port information to PDAs and touch screens.

A River Runs on It

Last year, Amazon.com put a number on how much Linux is saving the online retailer: $17 million in the third quarter alone. The savings accounted for a 25% reduction in the company's IT expenses for the quarter. The bulk of that came from buying Intel boxes instead of RISC-based Unix boxes. Amazon migrated 92% of its servers from Unix to Linux in less than four months with the help of outside vendors.

Spreading Like Caviar on Wall Street

Most major Wall Street firms are investing heavily in Linux to carry their computing loads because of Linux's inherent stability and phenomenal price point. Every one of the approximately 50 members of the techie-driven financial area users group had a Linux prototype running by 2001.

On the financial services side, E*Trade has made a significant shift to Linux. E*Trade began pondering Linux in early 2001 after considering the potential savings from moving to Linux on Intel-based servers, but it deemed the move risky. When IBM, HP, and others began making significant investments in Linux (full disclosure: both IBM and HP are members of OSDL), E*Trade decided the support infrastructure was in place. In 2002, the company bought 160 Intel-based servers and converted two-thirds of its data center away from Unix to Linux. E*Trade figures the switch saved about $13 million in maintenance, depreciation, and software licensing costs. When you run the numbers — $4,000 Intel servers versus $200,000 Unix machines — it's easy to see why.

Big Fish in a Big Pond

The Unilever Group, a consumer products company that has been running its global IT infrastructure across 80 countries on different versions of Unix, is now committed to a full migration to Linux over the next decade. Unilever's IT department not only wants a cookie-cutter approach globally with its data infrastructure, the company also wants to lower operational costs and realize the performance gains from Linux that it has already seen by running Linux on its Web, e-mail, and proxy servers and its firewall applications. Unilever hasn't yet projected total cost savings, but the company is encouraged by the performance gains and cost savings it has seen so far. For example, the company's firewall servers are running three times faster on Linux at 40% of the former operating and hardware costs.

Brain Food at the High End

DaimlerChrysler is the first auto maker to achieve supercomputer performance with Linux-based workstations. The Chrysler Group figures it saved 40% as compared with existing solutions on the market. The Linux cluster is based on 108 workstations powered by dual Xeon processors.

Steps to Linux Implementation

Now that we have seen some examples of the performance gains and cost savings that companies achieve by switching to Linux, it's time to look at the migration process.

If your organization has decided to implement a Linux project, it is critical to make sure that your IT people want to succeed with Linux. You may have a guy who is playing with Linux at home but is invested in staying with the incumbent vendor. A Windows NT or Unix IT manager is going to find reasons to stick with the current platform. That's always the case with disruptive technologies. And make no mistake, Linux is disrupting the status quo all over the data center.

What to do? The following considerations can help minimize the disconnects that have plagued implementations by those who came earlier.

Think about whom you're going to work with to make this successful. Know where your incumbent vendors are with Linux and discover what their future plans are. If this project is not being implemented from scratch and you're going to be using third-party support, turn to your established vendors.

Recognize that moving to Linux is no different than any other project. As long as you treat it that way, your chances of success are going to be that much better. Some IT departments have forgotten that, with Linux, the open source methodology may be different, but the company's internal processes and procedures should not be abandoned. Ensure that you have a solid operations plan for deploying, implementing, and administering your mission-critical solution on Linux. Stick to it.

Determine what pieces make sense from current suppliers and what pieces make sense from a different supplier. Your organization could be using a fully implemented Linux stack without ever going to the open source development community — IT people can download open source products from their respective Web sites or purchase them from vendors such as Red Hat, MySQL, SuSE, and others. That is a very reasonable approach.

Remember that the make versus buy decision works exactly the same for Linux as it does for proprietary software implementations. Linux is a modern operating system. As such, any implementation on a Linux platform can be treated in much the same manner as an implementation on a system running AIX or Solaris. The real differences between Linux and Unix systems lie in the access to and distribution and licensing of source code. An IT organization's well-established implementation processes will work with Linux and do not require any change.

Look for other companies already using your mission-critical application on Linux. As we saw earlier in this chapter, the number of successful implementations is growing rapidly. It is increasingly easy to find other people who have successfully implemented a similar application.

Linux Swims Further Upstream

Moving from the data center to the telecommunications market, the challenges are different. Most customers who buy a telecom switch don't really care what OS is inside; they're more interested in the final price point and reliability. It won't be an issue of wanting Linux because of features a, b, and c, but because Linux provides good bang for the buck.

Even though many network equipment providers have spent millions developing their proprietary systems, they are now realizing that it doesn't make any difference what the OS is as long as a switch meets reliability standards. Network providers might specify features and level of functionality but not the OS, and for this reason we foresee a mass migration of Linux into telecom switches.

Linux Kernel Sans Casing

Before that migration occurs, Linux has to beat the bad rap it has received from some. In some circles, Linux has a reputation as being raw. This perception results from the transparent nature of open source code development. For those who have been inside a proprietary kernel build, the process is no different. Open source development is a bit unsettling to watch, but the end result is high-quality software code. Decisions about kernel code are based on technical merits rather than which section has the manager with the biggest club. At least you get to see what's inside your kernel sausage.

Higher in the Food Chain

The idea that Linux is for bottom-feeders is a myth. The reality is that Linux can be the basis of a very robust platform for most applications. The experiences of the enterprises discussed above have been repeated many times. There have been numerous successful Linux implementations that provide real business value in tandem with reduced costs and risks. Linux is a fully functional, well-architected, and well-implemented operating system with support contracts available from many vendors.

Of course, Linux was initially developed on the desktop and for simple servers because that was what the independent community of open source developers owned or had access to. As Linux started sneaking in the back door of the enterprise, it leaped from hacker to executive. The ROI spawned increasing enthusiasm, and it became clear to a group of IT providers that developers needed access to enterprise-scale systems. Thus, OSDL was conceived and born.

Where Penguins Go to Eat

Linux supporters launched OSDL in 2000 as a global industry consortium — governed by an independent board of directors — dedicated to removing barriers to the adoption of Linux in the enterprise. The lab provides resources and expertise to open source developers who are building data center and carrier-grade enhancements into Linux. It also facilitates developer access to intellectual capital, product, and technology resources.

OSDL has invested more than $8 million in systems and services, hosting a state-of-the-art server farm with more than 120 systems available to open source developers at no charge. To date, we've sponsored more than 8,000 performance test runs of Linux code on OSDL's automated Scalable Test Platform environment. OSDL occupies 14,000 square feet of lab space in Portland, Oregon, USA, and Yokohama, Japan.

The result is that today any business can confidently implement a Linux-based solution or move an application to a Linux platform from any of the major vendor platforms.

CHAPTER 2

Open Source Java: *Fortune* 500 Systems at Two-Guys-in-a-Garage Prices

by Ganesh Prasad

Open source has the attractions of liberal and affordable licensing, quality products, and an active developer community. Java has a powerful language and platform model, industry acceptance, and interface specifications for virtually every kind of service that an enterprise could require. It hasn't taken long for a quiet marriage to take place. The phenomenon known as open source Java has now taken over a major segment of the software development world. With implementations of all the Enterprise Java specifications and development tools available for the price of a download, a fundamentally new disruptive force has been unleashed. Can you afford not to take advantage of it?

The Anatomy of an Enterprise App

Ingot Investments (not its real name) is a small fund manager in Sydney, Australia. Its 10 employees have a modest office but ambitious dreams. They plan to be one of the big fish one day, and they have ideas that will take them there. They also have very specialized requirements from their portfolio management system, requirements that the Windows PC-based software they had back in 2001 was unable to meet.

The folks at Ingot spent considerable time and effort looking for a package that would do what they wanted. It had to be multiuser with a single shared database. It had to be securely accessible over the Internet by partners of the firm who worked from home. And there was a host of specialized queries and reports.

Finding nothing on the market that the company could afford and that did what it wanted, Ingot finally hired a freelance developer for a couple of months. Working with freely available tools, the developer built a Web technology–based application that was exactly to Ingot's specifications. They were pleasantly surprised that they didn't have to spend very much money, and all the money they did spend went toward the developer's fees. It was win-win for both parties, because there were no software license fees to reduce the value they derived from the transaction.

The technology? The developer chose Linux as the server operating system, PostgreSQL as the database, and Jakarta Tomcat as the Web/application server. He wrote the entire application logic in Java as a set of JavaServer Pages, with client-side interactivity provided by standard JavaScript. Ingot employees access the application through their browsers, and those working from home have secure VPN access. The application has been in production for more than a year, and the partners are absolutely thrilled with their decision to build rather than buy. They would never have found an exact fit from any off-the-shelf product, and building a new one on top of a mainstream commercial platform (such as ASP, Visual Basic, Windows 2000 and SQLServer, or WebLogic and Oracle) would have been prohibitively expensive. Open source software saved the day.

What's important to note here is that the application in question is at the heart of Ingot's operations. It's the company's bread and butter. This is a mission-critical enterprise application, nothing less. And it runs flawlessly on an open source platform. It is not a low-end solution that Ingot will soon outgrow and have to throw away. As the company grows, the people at Ingot have the option to migrate parts of it to a three-tier architecture using EJBs. They can continue to avoid software license costs by using an open source application server such as JBoss. Being conservative and cost conscious, Ingot's managers are now absolutely sold on the value of Java and open source software.

Along the way, they have noticed something strange and outside their experience of years of running Microsoft software: the application hasn't crashed once since it went into production.

Open Source Java: The Dark Horse

In enterprise development circles, there are currently two major technologies that are battling for the minds (and dollars) of developers. One is Sun's J2EE and the other is Microsoft's .NET. Both are frameworks of technologies that seek to provide everything that developers need to build extremely large and capable systems for the most demanding applications. Arguments abound as to which is "better," and there is no end to the speculation about which will finally win.

However, this is not a two-horse race between J2EE and .NET. This race is three-way, between .NET, commercial J2EE, and open source J2EE. It is my opinion that open source J2EE (and open source Java in general) provides so much value compared to either of the other two contenders that it will end up being the most widely adopted technology in the near future, if it has not already achieved that distinction.

In the following sections, I will look at the various components of open source Java (open source J2EE in particular) and show how organizations large and small can derive value from the technology. For it is not merely minnows like Ingot Investments that use this technology. Giants like FedEx do, too.[1]

The Platform

J2EE is a set of specifications from Sun Microsystems that define the various components of an enterprise system through standard abstractions. For example, the Java abstraction for any database is provided by JDBC (loosely thought of as Java Database Connectivity). Vendors provide implementations that conform to these specifications, and users can use any competing implementation without having to change their applications, because the interfaces the application sees remain the same. In our example, any application an organization builds only knows about JDBC, not the actual database behind it. It could use Oracle, DB2, SQLServer, or PostgreSQL — the application doesn't care. This gives users and user organizations a great deal of choice and flexibility — and, consequently, negotiating power.

One of the fundamental value propositions of the J2EE platform is the notion of a "container," a server environment that takes care of many boring housekeeping tasks that developers would otherwise have to write themselves. J2EE defines two kinds of containers: Web containers and application containers. The latter are usually referred to as EJB containers, which I will explain shortly.

When a developer builds an application and "deploys" it to a container, that container takes over the running of the application and enforces security access rules as defined during the deployment. It also takes care of performance and robustness issues through the nature of its design.

Web containers such as the Apache Jakarta Project's Tomcat are lightweight servers that work well for smaller and less complex applications that are accessed only through the Web. EJB containers like JBoss, on the other hand, can interface to Web containers, but their own logic is independent of the channel of access. They tend to focus on the core business logic of a distributed application rather than presentation through the Web medium. The kind of

[1]See www.nwfusion.com/news/2002/1118fedlinux.html?net.

component that does such a piece of business processing is called an Enterprise JavaBean (EJB). Table 1 shows some commercial and open source containers.

The open source implementations (especially in the case of EJB containers) can save organizations significant amounts of money compared to their commercial equivalents because commercial EJB containers cost tens of thousands of dollars per CPU.

Consider the Norwegian Post Office, which deployed a distributed application at more than 200 outlets using the open source JBoss application server. Had it used a commercial alternative, the license fees alone would have run into millions of dollars, rendering the project infeasible or requiring the agency to make compromises in order for the project to fit its budget. Open source J2EE allowed the Norwegian Post Office to design an appropriate architecture for its application, unconstrained by commercial licensing costs.

Table 1 — J2EE Containers and Implementations

J2EE Container	Commercial Implementation	Open Source Implementation
Web container	WebLogic Express, "lite" versions of most EJB containers	Tomcat, Jetty
EJB container	WebLogic, WebSphere, Oracle 9*i*AS, Sybase EAS, Sun ONE	JBoss, JOnAS, OpenEJB

The Interfaces

Applications of more than basic complexity generally need to interface to other systems. Rather than having to rewrite parts of the software any time an interfacing system is replaced, Java abstracts out the external system through a Java language interface. Java programmers only refer to these external systems through the standard Java interfaces.

JDBC is just one example of an abstract interface specification that allows users to mix and match implementations. Table 2 shows different J2EE interface specifications and provides examples of actual implementations. The interface concept of J2EE has allowed all parties to interoperate without significant effort.

Table 2 — J2EE Interfaces and Implementations

J2EE Interface	Commercial Implementations	Open Source Implementations
JDBC (Java Database Connectivity)	Oracle, Sybase, DB2, SQLServer	PostgreSQL, MySQL
JavaMail	Any POP3- or SMTP-compliant mailserver	Sendmail, qmail, Postfix
JNDI (Java Naming and Directory Interface)	Any LDAP server	OpenLDAP
JMS (Java Message Service)	WebSphere MQ, SonicMQ	JBossMQ

The Development Tools

Traditionally, development tools have focused on graphical interfaces and ease-of-use features such as drag-and-drop. There is a special category of software called integrated development environments (IDEs) that bundle components such as an editor, compiler, debugger, online help, and more. However, a new generation of open source tools is now delivering such tremendous productivity and quality assurance advantages that many commercial IDEs are forced to support them. One of the foremost tools in this category is Ant (which I discuss in detail later).

There is a sense that with this support, there is less reason to use all the native features of the IDE. The IDEs have been "hollowed out" by open source. Therefore, the natural next question to ask is about the residual value of the IDE itself. Since some of them cost a few thousand dollars per developer, they clearly deliver less value than an open source IDE that has similar support for Ant. Open source is reducing the perceived value of commercial IDEs (see Table 3).

Ant is a tool that is now *de rigeur* for any kind of Java development. Ant can control the various activities of the build/deploy cycle. It is so powerful and elegant that a readily understood Ant script can compile, unit test, package, and deploy an application to a container, all with a simple command or mouse

Table 3 — Java IDEs

Commercial IDEs	Borland JBuilder, IBM WebSphere Studio Application Developer, Oracle JDeveloper, IntelliJ IDEA
Open source IDEs	NetBeans (donated by Sun), Eclipse (donated by IBM)

click. A regular fixture on many projects these days is JUnit, a tool used to write unit tests. Ant can control the compilation and running of the tests in exactly the same way that it controls the compilation, packaging, and deployment of the application. Ant was a relative unknown in 2001, but it caught on rapidly in 2002.

History appears to be repeating itself with a tool called XDoclet, which began to gain quiet acceptance in 2002 and seems likely to become standard in 2003. XDoclet generates much of the code and "deployment descriptors" specific to each container. It is possible to eliminate a lot of drudgery and redundancy in the way EJB applications are currently written. XDoclet can generate code for many aspects of a J2EE application, not just EJB interfaces and deployment descriptors. XDoclet can generate Web tier deployment descriptors ("web.xml"), Web framework configuration files (Struts), JMX (Java Management Extensions) interface code and descriptors, JDO (Java Data Objects) metadata, object-relational mapping code for the Hibernate tool, and so on. If they use XDoclet wisely, developers can improve code quality and reliability while also dramatically improving productivity. XDoclet is one of the most important Java development tools to emerge in recent years, and no development shop can afford not to incorporate it into their standard development process.

On the Web side, some interesting frameworks have sprung up to make development more structured and less error prone (virtually all of them open source). The foremost among these is Struts. Struts lets developers build complex screen flows with relatively simple mappings. Webwork and Maverick are other frameworks.

Table 4 describes the open source tools that you are likely to need on a project. Notice that unit testing tools and frameworks are part of the set. In combination with an open source IDE like NetBeans or Eclipse, these tools may be all you need to build an enterprise-class application in a highly productive and quality-assured manner.

Table 4 — Open Source Java Development Tools

Open Source tool	Function
Ant	Build and deploy tool
CVS	Version control tool
XDoclet	Interface class and deployment descriptor generator, extensible with many features
JUnit, JSPUnit, HttpUnit, Cactus	Testing frameworks and tools
Struts	A structured framework for JavaServer Pages
Velocity	A forms templating engine for JavaServer Pages
Maven	A project management and dependency-checking tool

The Processes

Finally, there are newer methodologies that, although independent of open source and Java, have grown up around Java projects. Extreme Programming (XP) is a technique that, among other things, teaches testing before coding. Tools like Ant can help to ensure code quality by running a suite of unit tests automatically with each build. This enables XP virtues such as “fearless” modifications, because the tests catch anything that breaks as a result of changes made.

Of course, when a team is making lots of changes, version control becomes that much more critical. Concurrent Versions System (CVS) is an excellent version control system that uses an optimistic locking policy. (Optimistic locking assumes that no two developers will change the same piece of code at the same time. If they do, the one to check in changes last must resolve conflicts.) Provided developers are careful about the areas they work on to avoid conflicts, this optimistic locking policy should be acceptable. There are good commercial alternatives to CVS that employ a more conservative pessimistic locking policy (e.g., PVCS, Microsoft Visual Source Safe), but they obviously cost more money.

As for documentation, using Javadoc for application documentation is good practice, as usual. It so happens that XDoclet relies on Javadoc-style comments to generate metadata files and dependent classes. Since the Javadoc mechanism has to be used when XDoclet is part of the development process, it is very little additional effort to add regular Javadoc comments describing the application’s classes and methods. The organization’s procedures should make this a requirement.

Conclusion

Open source Java is here to stay. Because it offers the high-quality tools and technologies that a project team requires, it is already becoming the standard way to build applications.

One residual problem with open source — even with all these excellent tools — is the issue of mutual compatibility of the various versions of independently developed products. Version 1.1 of tool A may only work with version 2.3 of tool B. If you happen to use version 1.2 of tool A, must you also upgrade tool B to version 2.4? Isn’t it too much trouble having to keep track of these dependencies? Any time and money an organization may save by going the open source route can be quickly squandered if developers are forced to chase down obscure bugs caused by version incompatibilities.

Some companies have recognized the business value in bundling a consistent set of all required open source project tools and charging for support, maintenance, and upgrades. EJB Solutions, Inc., for example, sells a copy of all the above software and offers e-mail support as well as some tutorials. The company performs the necessary testing to ensure that the versions of the open source toolkits in its bundle are compatible with each other. For a nominal fee, developer organizations are freed from infrastructural headaches and can get on with the task of building applications.

Whether your organization is large or small, it is highly likely that you will derive far greater value by using open source containers, IDEs, and other development tools than by going with a proprietary alternative, whether that is commercial J2EE or .NET. So what are you waiting for? Give open source a spin!

CHAPTER 3

Come Together, Right Now: Eclipse and Open Development Tools Integration

by Marc R. Erickson

Paradigms are changing quickly for software developers. It may be hard to believe, but commercial-quality software that has already demonstrated superior utility and business impact is now coming from the open source movement. This is occurring within a developing ecosystem of tool producers and consumers that lets everyone focus on what they do best, competing on the merits of their implementations of technology. It's based around an extendable micro-kernel-style integration platform called Eclipse.

Software developers who create tools are probably unique in the computing industry. These individuals have the ability to change the very essence of the development process with their tools. There are dozens of popular languages, hundreds of deployment platforms (embedded through enterprise servers), and numerous vendors all competing with labor-conserving offerings. Ever since language constructs were considered for programming computers (a great leap forward from raw machine or even assembler coding), the integration of these tools has improved exponentially.

As with many examples of technology development, software tooling has progressed through generations of advancement, starting with the command line compiler, progressing to vendor-associated integrated development environments (IDEs), and on to platforms that embrace the complexity of middleware and deployment resources like Web services servers. The most important advance is in how we're now approaching tool development and the new open culture that complements that approach.

Talkin' 'Bout My Generation

Eclipse represents the next generation of tools technology. Unlike the majority of other development offerings, it embraces an architecture that didn't originate in a single language or middleware deployment environment. Eclipse is not a "suite" of tools, but rather a well-implemented integration framework that vendors plug into, reusing common components. Java development tools (JDT), C/C++ development tools (CDT), and many more offerings — both open source and commercial — implement IDEs and other extensions by plugging in to the Eclipse Platform. Thus, Eclipse doesn't restrict you to a preferred vendor, language, or platform but seeks to embrace all through a neutral and open approach. This is truly "open kimono" development tools integration.

How can new development environments best support the coming generation of end-to-end computing technologies? It's clear from our experience deploying the Eclipse Platform that open source collaboration is the basis of the best environment. For example, tools supporting deployment and maintenance project phases are often missing from traditional IDEs. Technology for functional and load testing often comes from developers outside the traditional "tower" environments established by proprietary technology vendors. Most IDEs get you as far as unit testing the code and checking it in to repositories, but what about ongoing environmental testing, injection of changes into production environments, and collaboration with operations on problem management? Open source lowers barriers to entry for tool providers working on new technologies. Providers can freely reuse what they need from open source projects and extend into the deployment facilities that they support.

Many providers are now making a "plug-in" version of their technology for both Eclipse and other development environments. In doing so, vendors acknowledge the importance of preserving a developer's choice of workbench and integration technology. Open source tools that preserve a developer's choices need to be freely transportable to popular development platforms. (In the case of Eclipse, these include Windows, Linux Motif, Linux GTK, Solaris Motif, HP-UX Motif, AIX Motif, Mac OS X, and QNX Photon.)

It's equally important to address a broad variety of deployment platforms, from the smallest embedded computer to the largest mainframe server and everything in between. This is critical to projects that implement services spanning these environments.

It's only in a vendor-neutral and language-agnostic environment that a level playing field for the entire development ecosystem can be created. Both as contributors and beneficiaries, members of open source communities like Eclipse's are establishing a clear advantage through this approach. In such an environment, it's more valuable to interoperate with the developer's choice of workbench and tooling than to try and force loyalty to an isolated solution. Developers choose the most appropriate tools for a project based on their merits. To be truly international, tools must work in the local spoken language and allow development in a wide variety of computer languages. With so many choices and technologies, it's unlikely that one individual or vendor can be the best and cover all the bases.

Providers and consumers make choices and then live by them. A community forms around the chosen tools. That community contains specialists who can often execute in their area of focus better than the competition. Such a community can form under the rigid rules of contracts, partnerships, and agreements, or it can form freely in open source. In the case of Eclipse, we've clearly seen that community form faster, to greater depth, and with better participation under the meritocracy-based culture of open source. Besides, does just another proprietary development tower make sense today?

Segregation Never, Integration Now!

The IDE has been the arena in which tools vendors competed on issues such as GUI look and feel, feature/function, and the partnerships that the vendor could establish. By their very nature, these partnerships have proven to be as limiting as they are effective for typical project developers. Often associated with specific Web services or middleware positioned through development tooling, they either limited the range of choices that a developer had or led to an integration black hole in which project teams either developed their own solutions or coped with manual processes over the life of the afflicted project. From time to time, for business, technology, or process reasons, development teams change the tools that they use. When that happens, all of the investment in making development tools work together is not only lost, but also demanded again by the new environment. This considerable hurdle greatly impairs a team's flexibility.

If our development process is to be useful in a broad range of software engineering efforts, seamless integration of tools from vendors is a must. This integration cannot be limited to just the vendors that have business or technology partnership relationships. Clear trends are now emerging for the development culture itself. The number of freelance developers moving between organizations from job to job has skyrocketed. Development centers specializing in various aspects of computing are emerging in such far-flung places as St. Petersburg, Russia (animation); Bangalore, India (embedded computing and user interfaces); and technology centers in China (embedded computing). End-to-end project development and deployment processes demand tools that work well together regardless of development or deployment platform, developed object contents, the location of team members, or the native language a developer speaks.

What’s needed is a platform that delivers a true Integrated Everything Environment (I*E, or “I-star-E”) that can serve as the framework upon which to build up the tools used by a geographically dispersed, Internet-linked development team. The I*E needs to integrate tools that address the needs of all phases of development and project deployment from requirements through modeling, code development, debugging, environmental testing, deployment (to multiple platforms), and maintenance. The I*E must establish the level of interoperation needed to integrate tools that implement the Object Management Group’s (OMG) Model Driven Architecture (www.omg.org/mda) and the tools needed for studying ever-changing deployment issues. Aspects of project component testing now must range into end-to-end computing scenarios that change over the life of a project as new Web services servers and Internet service providers (including wireless connection services) are added to the deployment mix.

Next-generation computing technology will integrate smart connected devices with Web services currently deployed or in development. In this environment, smart devices that are aware of their environment and in control of their surroundings through sensors and actuators will become omniscient assistants that make everyday tasks easier. With the addition of low-cost wireless tablet or handheld computers as user interface–rendering devices, services based on smart devices will abound.

This environment demands the collaboration of computing technology specialists who understand the back-end services, mass server deployment (often serving the demands of millions of concurrent devices), networks, embedded devices, and middleware that make on-demand pervasive computing possible. Projects will surely require the skills of many of these specialists in both development and deployment phases. Often the skill will come from contractor/subcontractor relationships between separate companies or different departments in large organizations.

Preserving choice in this environment is key to success. For example, the different companies or departments involved in a project might make use of different team repository systems for source code control and version/configuration management. Languages used to deploy portions of the project in different runtime environments may vary from Java to embedded processor assembler and the variety of metadata used to define and configure middleware components. How can a development platform best accommodate this multilanguage, multiplatform, multivendor world?

For the Times They Are A-Changin’

Three dominant themes have appeared in the market for enabling technology in 2003-2005: integration, flexibility, and maximizing return on existing technology investments. New approaches to business process management are leading project architects to rethink the position of cyclic development processes in relationship to testing, deployment, and maintenance in cases where worldwide teams need to be coordinated. This is starting to include aspects of team collaboration tools (like instant messaging and collaboration databases such as Lotus Notes or Microsoft Exchange) and integration of traditional systems management tooling for deployment testing, production injection, and problem management/source identification/resolution.

The end-to-end tasks associated with development now include requirements gathering and tracking systems, function point analysis, modeling, model driven architecture, load testing, and hot-code injection of maintenance or functional updates. No single company has yet delivered the best-of-breed product in all of these development and deployment areas. No single company is likely to do this in the future. In fact, our computing industry is at worst an oligopoly and at best a strongly competitive marketplace in almost all of the aspects of computing. Technology vendors (be they software, middleware, hardware, or network services vendors) all compete based on feature, functional coverage, quality, and business merits. In this marketplace, partnerships often create artificial business relationships that do not lead to well-wed technology. The thing that protects a development team’s flexibility and choice in technology is an open and level playing field for all the competing vendors.

In this environment, open source has proven the great leveling agent. Again and again, superior technology has risen from the devoted interest of individuals and companies collaborating on open source project–based solutions to common problems and common interests. Access to shared source code ensures that nothing takes place through secret closed interfaces available only to a primary vendor and its favored few. In this meritocracy, developers go head to head with alternate technical solutions to problems, and even a neophyte has ubiquitous access to the assets of the project. The next creative and innovative solution is as likely to come from a college freshman as from a corporate think tank. This is the essence of openness. The power shifts from possessing the asset to understanding it. It shifts from controlling behavior to openness and collaboration.

In a recent move, the Palo Alto Research Center (PARC) transferred a DARPA-funded research project to Eclipse when that funding ended. In some ways, this recognized the essentially democratic nature of placing technology implementation within the framework of an open source meritocracy. The implementation of AspectJ language tools, which enable a new approach to the development process called "aspect-oriented software development," now rests with a formal subproject within the Eclipse Technology Project.

Plugged In

IBM participates broadly in open source projects, with more than 1,000 IBM software projects already released into open source. One of the largest was the release of the Eclipse Platform in November 2001. An open universal tools integration platform, Eclipse allows tool providers to freely "plug in" to the framework, extending platform facilities. More than 89 million Web hits, more than 1.5 million visitors, and more than 6.9 million download requests have been logged at the Eclipse Web site (www.eclipse.org). Eclipse is becoming the open source I*E, with new tools projects being launched now to provide components that support Model Driven Architecture[1] and a full-cycle test and trace integration framework. There is already a large number of university-based research projects that work with and reuse Eclipse technology, as well as over 230 commercial and noncommercial efforts to create Eclipse plug-in tools.

Eclipse is demonstrating the industry-wide value of approaching the implementation of the next generation of tools through a common tools integration framework. The Eclipse open source projects recognize standards where they exist, providing a collaborative environment for creating useful reference implementations of the technology that are made available without charge. Where standards don't exist, Eclipse demonstrates an approach to creating effective solutions that the industry can share.

Vendors that are freed from the costs of developing and maintaining the common components they reuse from the Eclipse distributions are now working to implement the next generation of integrated tools. They are focusing on specific areas of expertise and exploring approaches that will lead to more efficient project development, deployment, and maintenance. Many of these offerings, including a broad range of supported commercial products, are available to developers today.

[1]The Eclipse Modeling Framework (EMF) implements a subset of the standard OMG Meta-Object Facility (MOF).

As tools and middleware vendors and technology standards organizations approach next-generation solutions to support service-oriented application architectures, many have chosen to become supporting members of the Eclipse consortium:

- Borland/TogetherSoft
- IBM/Rational
- MERANT
- QNX
- Red Hat
- SuSE
- Fujitsu
- Serena Software
- Sybase
- Hitachi
- Instantiations
- MontaVista Software
- Telelogic
- Trans-Enterprise Computing
- ETRI (Korea)
- HP
- MKS Software
- SlickEdit
- AltoWeb
- Catalyst
- Flashline
- Parasoft
- OMG
- SAP
- Teamstudio
- TimeSys
- Ericsson
- Fraunhofer FOKUS (Germany)
- LogicLibrary
- QA Systems
- SilverMark
- Oracle
- Scapa Technologies

CHAPTER 4

Making Open Source Make Money

by Luke Hohmann and Michael Olson

Let's listen in on a recent conversation with one of Luke's clients, a vice president at a *Fortune* 200 company that provides an enterprise-class software application for managing geographically distributed mobile workers. Ned's team had dominant market share in its core market and wanted to conquer new markets. Not surprisingly, these new markets required new business and license models. Could open source licensing be part of the solution? Using the successful licensing models of Sleepycat Software in Berkeley, California, as a guide, the answer turns out to be a resounding "yes."

A Skeptical VP

"Luke, I like your ideas about how we can improve our pricing and business models, but I don't see why you think some of the utilities associated with our application should be offered under an open source license. Sure, open source works great for hackers who are making free utilities, but I'm running a commercial software business. Open source can't make money. I'll be out of business if I don't charge my customers license fees for everything that I do."

We were sitting in Ned's office. He had called me in to explain some of my ideas regarding open source and how it might be used in a project to create a set of utilities that would monitor the use of Ned's software.

"Ned, I understand that the business and licensing model I'm proposing for these utilities is different from your normal business and licensing models, but the market research shows that your customers simply don't want to pay your standard licensing fees. By offering these utilities using an open source model, I think that you'll actually *increase* total revenue by driving sales of your core software and services."

"But open source can't make money. I've read the GPL,[1] and its "viral terms" mean that I can't base my offering on that. I'd have to open source my entire application, and I'm not willing to do that."

[1]GNU General Public License.

"You're right about the GPL, but that's not the only way to license open source software. There are several variants on open source licensing, and there are companies that do make money using open source models — even for their core products. Sleepycat Software, for example, has taken an innovative approach to open source and created a sustainable, profitable company. I think some of the ideas they've pioneered can help you make more money."

What's your position on open source? Like Ned, do you believe that open source simply can't make money and is therefore inappropriate for for-profit organizations offering software-related products and services? If so, read on. You'll gain a better understanding of how business models, license models, and intellectual property rights can work together to help you increase revenue, lower support costs, and open new markets through open source licensing.

First Things First

Your *business model* is the manner in which you charge customers for your products or services — the way you make money. Software products and services are offered through a *license model* that defines the terms and conditions (or rights and restrictions) that you grant to a user and/or customer of your software as defined by your business model. The degree of freedom you have to create business and license models that achieve your objectives are governed by intellectual property rights, including the rights you've obtained from technology that you incorporate (through in-licensing) as part of your total solution.

The most common software-related business models make money by:

- Providing unfettered access to or use of the application for a defined period of time (e.g., a perpetual or annual license)
- Providing one or more services that are intimately related to application operation and/or use (e.g., ISP and e-mail providers, or consulting and integration services)
- Charging for a transaction, a defined and measurable unit of work
- Metering access to or use of the application or something the application processes (e.g., charging per CPU or concurrent user)
- Charging for the hardware the application runs on, and not the application itself
- Charging a percentage of the revenue obtained or costs saved from using the application

In theory, any of these business models can be used by vendors offering their software under open license models. In practice, some combinations don't make good sense, and time-based licenses and associated services have emerged as the most prominent open source–related business models. For example, while you can't "sell" Apache, you can certainly base your business model on installing, configuring, and operating Apache-based servers.

I continued, "Sleepycat makes Berkeley DB, an embedded database used in a variety of products ranging from routers to directory servers. You can use Berkeley DB free for internal use, even if you're using it to support commercial services. Several ISPs do this. The implications of their licensing models kick in when you ship a product using their software. Sleepycat provides their software for no charge, provided you also make the complete source code for your application freely available at no charge. This is similar to the GPL."

Ned jumped in (as VPs at *Fortune* 200 companies often do), "See! By using GPL, they prove that you can't make money at open source."

"Not so fast," I replied. "That's only half of the story. Several years ago, when Sleepycat was founded, they realized that for-profit companies that wanted to keep their source code proprietary couldn't agree to these terms. So they created a separate license for commercial use. This license agreement looks and acts identical to other annual license agreements from other vendors."

"Let me get this straight. If you want to use Berkeley DB in a product, you obtain an annual license for commercial use. This license would be like the licenses from other vendors that provide proprietary software — say, a GUI widget or middleware — that's integrated into your application."

"Yeah, that's the gist of it. If you're providing your software under an open source license, so is Sleepycat. If you're not, Sleepycat acts like any other supplier of in-licensed software."

"And they make money at this?"

"Yes, they do. They're profitable. And their revenue distribution matches what you'd expect for a provider of core libraries. About 75% of their revenue is from licensing, and the rest is from associated services."

Ned turned his head to left and looked at the ceiling. I knew this meant he was thinking, so I stayed quiet. It wasn't long before he broke the silence.

"If Sleepycat is making money under a traditional licensing model, why offer an open source version?"

Ned, like many others, suffers from what I refer to as "Model T" business and license model strategies. These people create few business models and expect each of their target markets to adapt to them. Moreover, they define relatively rigid licensing models, failing to realize that, within a business model, market segments will pay to obtain certain kinds of rights (like the “right to upgrade”) or remove certain kinds of restrictions.

In the case of Sleepycat, it is clear that customers who wish to use Berkeley DB in commercial products are willing to pay for this right, and by offering two distinct licenses, Sleepycat can meet the needs of a diverse market. Sleepycat pulls this off because they own all of the intellectual property associated with Berkeley DB and because Berkeley DB is a library that must be linked with their application. Although Ned’s team has created a few alternative business models, he is stuck in thinking that the only kind of business and license model that makes sense is one where all source code is private and you must charge for what you release.

“Good question,” I replied. “There are several important things that Sleepycat gets out of this dual approach to licensing. One of the biggest is market share. By offering two licenses, Sleepycat can be used in more products. Consider the number of open source systems powering the Internet that need a high-performance embedded database. By allowing Berkeley DB to be used in these systems, Sleepycat gains share in markets that they wouldn’t otherwise be able to tap. A larger market creates more requests for enhancements, driving innovation. These enhancements benefit and even help obtain commercial use customers. The larger market also helps pollinate use of Berkeley DB as developers move between jobs and products.”

I continued, “Another benefit is that there are lots of people reviewing the source code, which improves overall quality.”

Ned looked puzzled. “How can Sleepycat offer a commercial use license for code they didn’t write?”

“What do you mean?”

“Well, if some developer looks at the Sleepycat source, finds a bug, and fixes it, don’t they own the fix?”

“Oh, I see. Sleepycat’s license terms are more restrictive than the GPL. In the GPL, changes that you make must be distributed to everyone. In contrast, the Sleepycat license requires that all rights to any changes you make to the source code be assigned to Sleepycat. They own all of the IP. In practice, this means that few people actually make changes to the source, but its open nature means that anyone can look at it. And they do, because we both know that you can use a library more quickly and effectively when you have the chance to peek under the covers.”

Ned turned his head to left again and began to think a bit more. “Well, you’ve convinced me that Sleepycat’s approach works for them. But I’m not sure I see why you think some of these ideas will work for us, since we’re already successfully using multiple business and licensing models. Besides, Sleepycat is a library that you embed in your application, and we're a complete system. How, exactly, will we benefit?"

The Business Benefit of Open Source Licensing

Ned's offering is a mission-critical, enterprise-class, client-server application for managing geographically distributed mobile workers. The system is complex (it has to be!), has been on the market for several years, and over time has evolved to include a mix of business models. The current business models include:

- A one-time perpetual license fee plus an annual maintenance fee for the server and client software applications
- Transaction fees based on the number of messages (transactions) transmitted to and received from mobile workers
- Hardware fees for the proprietary hardware provided to mobile workers
- Service fees for installing, maintaining, and integrating the system

Ned's current customers were happy with these business models, but prospects in newly identified markets found the variable nature of the transaction fees untenable. For these markets, we had created a new business model that removed the variable nature of the license fees, and we were preparing to roll this out. Customers in this market also requested a set of additional utilities to manage their use of the system, but they felt strongly that they should not have to pay additional license fees for these tools. The problem was that some of Ned's integration partners could also use these tools for their gain.

Despite the strong feedback from the market, Ned's product managers proposed offering these tools under an annual license to both customers and partners in order to protect corporate interests. I was concerned that doing so would inhibit adoption of the tools — a negative self-fulfilling prophecy. I was also concerned that developing these utilities might require more resources than originally estimated, and I was looking for a solution that could lower risk. I felt that a more creative licensing policy that incorporated open source ideas might be the answer.

Ned had asked another good question: How could ideas inspired from open source licensing meet his needs?

"Ned, before choosing a business or license model, let's step back and look at customer needs. We have evidence that, to really attack this new market, you're going to have to offer a new set of utilities for managing use of the system, but customers don't want to pay for this. What makes matters worse is that we're not entirely certain your customers know what they want in terms of systems management, so we could invest a lot of development resources in something that really doesn't provide a lot of value. Charging for something that may not meet customer needs doesn't sound good to me, and neither does investing a lot of development resources to try and 'get it right' for the first release."

"Me neither," Ned replied. "So, what do you suggest?"

"Let's try this. We'll create a minimal set of utilities — just a few of the ones that customers requested. They won't cost much to develop, but we'll have a concrete deliverable that your product managers feel will provide value. We'll offer these to customers in a business and license model that is partially inspired by Sleepycat. The utilities can be used by your existing customers internally, free of charge. They cannot redistribute these utilities. Service partners who wish to use or access these utilities will be required to pay an annual license fee. This is a bit different than Sleepycat's approach, but it's better for your business needs. Customers or service providers who want to enhance the utilities must send the enhancements to your development team and assign all rights in these enhancements to you."

"That's an interesting approach, Luke. My customers will get the utilities they want free of charge, and I can still make money with my existing service provider partners. But how does this mitigate the risks associated with my development costs?"

"I'm not sure. The development costs are the unknown wild card. Here are a few scenarios that I think could happen. One is that these utilities may not really be that valuable. So we invest a bit of money, customers are happy to have them, but, ultimately, few customers use them. This manages the risk by minimizing initial investment. The other end of the spectrum is that these tools become so valuable that customers become willing to pay license fees to support their ongoing development and maintenance."

"Well, what do you think will happen?"

"I don't think we'll see either of the extremes. I suspect that because these utilities will help customers lower their total cost of ownership, they will have an incentive to modify them to meet their needs, provided we give them a reasonable starting point to work from. We'll know this by the level of activity we see among customers after the tools are released. If we see an initial flurry of activity, then we know the tools are being used to drive down total costs. As customers start to realize sufficient costs savings, we should see less total effort. From there, you'll have to justify your ongoing investment by calculating the number of customers who used the availability of these utilities as part of their purchasing decision."

Ned's eyebrows furrowed. "But I'm still paying for the initial development — and quite possibly all of the ongoing development — in all of these scenarios. I thought the whole point of open source was to get other people to write code for you!"

I laughed. "That sounds great, Ned, but I don't think that's your primary business objective. My understanding is that you want to open a new market in a cost-effective manner. I think that offering these tools as open source creates a strong possibility that customers will extend them to meet their needs, to the benefit of everyone. Let's look at it from the opposite angle. If you go with your traditional closed source/annual license model, you guarantee that you'll have to foot the bill for all development costs on a portion of the system with unknown value. Using open source for this portion of the system means that you've created a greater numbers of options for you, your customers, and your partners."

In the end, Ned agreed to try releasing these utilities using the general business and license model described above. His team is still working through the details of exactly how this will happen along with the delivery of these tools.

Market Maturity Influences on Business Model

The maturity of your target market is one of the strongest influences on the selection and management of a given business model. In the early phases of a given market, business models should be chosen so that they can be quickly and easily understood, primarily because you may not be certain of the best way to structure the business model. You may find that your customers prefer an annual license to a subscription or that they expect discounts if they purchase in bulk. Moreover, despite the best intentions of the business plan, you might find that innovators and early adopters expect and/or demand special terms.

As the market matures, chances are good that your business model will need to become increasingly complex in order to serve the idiosyncratic needs of different market segments. I've helped several companies enter new markets by defining new business and license models. The principles of open source licenses and the creative applications of business models inspired by open source licensing models provide software vendors greater choice in how they reach target markets. I predict that we'll see open source licensing options and influence increase over the next few years.

Choosing a Business and License Model

Choosing a business and license model is one of the most challenging tasks senior decisionmakers face. It involves the issues raised in this chapter as well as a whole host of other factors, including but not limited to the business and licensing models offered by competitors (which may constrain you to existing market expectations) and corporate and/or environmental factors beyond your control (such as when another division does poorly and you need to find a way to increase short-term revenue). To help you through the potential morass of choosing a business model, consider these questions.

Who comprises the target market? What do they value? A crisp description of the target market and what they value is the first step in creating appropriate business and licensing models. If you're having trouble determining what your target market values, find out as much as you can about the problems you're trying to help them solve. Solving them is where the value lies.

What are your objectives relative to this target market? In an emerging market, you may wish to capture market share, so create simpler models. In a mature market, you may wish to protect market share, so create more complex models to provide flexibility. When attacking a new market from a strong base in a related market, consider the changes you'll need to make to the established models.

What is your business model? Pick one of the business models defined above and customize it to meet your needs.

What rights do you wish to convey? Begin by asking your legal department for a "standard" contract, then remove the set of non-negotiable rights and restrictions, such as warranties and indemnification. What's left, and what you can create from it, gives you creative power to meet market needs.

What is the effect of this business model on your software architecture? Work with the technical team to make certain that any business model you propose is appropriately supported. For example, if you're going to charge a transaction fee, your offering's technical architecture has to support it.

As you develop the answers to these questions, you're likely to find that the best way to reach a given target market will require a variety of changes to your current business model, licensing model, and software architecture. You'll have to rank order the changes that these require in all areas of your product so that you can reach the largest target market. The benefits will be worth it, as creating the right business and licensing model is good for you and your customers.

CHAPTER 5

Intellectual Property and Open Source: Copyright, Copyleft, and Other Issues for the User Community

by William A. Zucker, Senior Consultant, Cutter Consortium

Open source is not a new phenomenon. In the early days, when the computing industry's emphasis was on the sale of hardware, hardware manufacturers gave the software away without even bothering to copyright it. In general, software was not portable among hardware platforms, and the manufacturers were only too willing to have the user community find bugs and suggest improvements. As hardware became less important and software more so, the emphasis changed. Software became the focus of proprietary claims and protection. Ownership was rigorously guarded and lawsuits became commonplace as the software was perceived as the primary driver of the economic engine. Source code was regarded as proprietary, and only object or machine-readable code was to be circulated.

Swimming against this tide of corporate culture was the so-called hacker culture, which believed that source code should be shared. One of its primary exponents was Richard Stallman, who founded the Free Software Foundation and developed the GNU tools with an intent to create an open source system modeled on Unix. These GNU tools became widely popular and were ultimately used by Linus Torvalds to create the Linux kernel. The Linux kernel, packaged with the then-existing GNU-developed software, became the first Linux operating system.

Stallman also developed the GNU General Public License (GPL), although there were other existing models for open source licensing such as the Berkeley Software Distribution (BSD) license for a version of Unix that was developed at Berkeley. The main feature of the GNU GPL was that it required free distribution of the source code with the program and free distribution of any modifications to the program together with the source code. Thus developers and users were free to use, modify, and redistribute code so long as they adhered to the license. This essential notion of unprotected "free" distribution — as opposed to copyright's prohibition against distribution without the copyright owner's consent — has become known as "copyleft."

In order to understand the legal implications of open source and its pros and cons in your organization, it is necessary to place the open source movement and licenses in the wider context of intellectual property.

A Very Short Primer on Intellectual Property Rights

Trade Secrets

For many years, unless an idea was patentable, the primary protection for internal business data, confidential or proprietary information, and computer code was the common law doctrine of trade secrets. Generally, a trade secret might be considered any internal, nonpublished manufacturing know-how, drawing, formula, and/or sales information used in a trade or business that has commercial applicability and provides a business with some strategic advantage. As long as such information was (a) not published or disseminated to others who were not obligated to maintain its confidentiality,[1] and (b) maintained in confidence with the protecting organization, it could be protected as a trade secret.

The law of trade secrets thus recognized a business's ownership or proprietary interest in such information, data, or processes. There are, however, important practical limitations on the application of trade secret protection. First and foremost, for any product sold in the market, the law does not protect against a competitor seeing the product and then using it to figure out how to manufacture similar items. Competitors are therefore free to "reverse engineer" a product, provided the reverse engineering is done wholly independently.

The second caveat is that an organization has to prove not only that the information qualifies for trade secret protection, but also that it protected the secrecy of the information as required by the law of the applicable jurisdiction. This means that ownership is not a matter of record but of case-by-case proof, making enforcement of trade secret protection time-consuming and expensive later on.

Because of anxiety over the true extent of protection afforded to software under patent and copyright law, software programs were initially protected as trade secrets. Such protection has become increasingly problematic in today's society, where information technology and pressure for the free flow of information make confidentiality controls more difficult to police. Copyright law has now evolved to include computer programs.

[1]The need to protect the information from general dissemination is what, in part, has given rise to the practice of nondisclosure agreements.

The Copyright Act

Since 1964, the US Copyright Office has permitted registration of computer programs, although judicial decisions were divided on the applicability of the Copyright Act. In 1976, Congress passed the Copyright Act of 1976, which did little to resolve the ambiguity. Clarification was finally obtained in the Computer Software Copyright Act of 1980, which explicitly extended the protection of the copyright laws to software. Any type of work that can be fixed in any tangible medium is protected, even if the work can only be machine reproduced.

Copyright protection, however, does not protect "ideas" or "facts." Rather, it protects the particular expression of the idea. As we can see in the parallel proliferation of spreadsheet programs, the idea for the spreadsheet program cannot be protected, but the particular code that produces the spreadsheet can be. In order to qualify for copyright protection, the work must be (a) original, (b) fixed in a tangible medium, and (c) not just the embodiment of an idea. Once obtained, copyright protection grants to the copyright owner the exclusive right to reproduce and publish the copyrighted article. In 1990, Congress passed the Computer Software Rental Amendments Act, which added to the list of copyright infringements *the distribution of a computer program for commercial advantage*. Materials copyrighted after 1978 are protected for the lesser of 75 years from the date of first publication or 100 years from the date of creation.

For the purposes of this discussion on open source, the particular attributes of copyright law that are critical to software are the prohibitions on copying, distributing, or creating derivative works. Copying means just that, although there are certain permitted uses and "fair uses" allowed by the Copyright Act. Distribution is complicated slightly by the doctrine of "first sale."

Simply put, the doctrine of first sale limits copyright protection to the first sale of the item. Once the item is placed in commerce, subsequent transfers cannot be restricted. To avoid wholesale redistribution, licenses have become the means for restricting resale, because the license permits the licensor to restrict future use and resale. Indeed, one of the primary changes the Uniform Computer Information Transactions Act (UCITA) seeks to make is to deem all sales of software "licenses" so that the vendor can control subsequent use and (conceivably) what other software can be used in conjunction with the vendor's product, as well as how software can be linked.

Under section 106 (2) of the Copyright Act, the copyright owner has the exclusive right "to prepare derivative works based upon the copyrighted work." The Copyright Act defines a "derivative work" as:

> A work based upon one or more preexisting works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgement, condensation, or any other form in which a work may be recast, transformed, or adapted. A work consisting of editorial revisions, annotations, elaborations, or other modifications which, as a whole, represent an original work of authorship, is a "derivative work."

A derivative work is thus defined as an original work that could be independently copyrightable. To infringe the exclusive right of the copyright owner to prepare a derivative work, the infringer need not actually have copied the original work or even to have fixed the allegedly infringing work in a tangible medium of expression. It's enough if the creator of the "new" work had access to the copyrighted work and the new work is a "modification" of the old. The right to protect against derivative works that springboard from the original work is therefore an important weapon in the scheme of copyright protection.

Where software is concerned, there is a practical exception to the Copyright Act's prohibition against creating a derivative work. A lawful owner of a purchased license for a computer program may adapt the copyrighted program if the actual adaptation "is created as an essential step in the utilization of the computer program in conjunction with a machine and it is used in no other manner." 17 U.S.C. § 117. The adaptation, however, cannot be transferred to a third party. The right to adapt is, in essence, the right to modify or, in the language of the Copyright Act, to create a derivative work. Such changes can be made even without the consent of the software owner so long as such modifications are used only internally and are necessary to the continuing use of the software.

In 1998, Congress passed the Digital Millennium Copyright Act (DMCA) to address concerns raised by the Internet and copyright issues in the context of our increasingly technological society. The DMCA creates a civil remedy for its violation, as well as criminal penalties starting after October 2000. One of the purposes of the DMCA is to protect the integrity of copyright information. Removal of copyright notice, or distribution knowing that such copyright has been removed, is now actionable. Both civil and criminal remedies also now exist if one circumvents "a technological measure that effectively controls access to a work protected" by the Copyright Act. Thus, efforts to circumvent access limitations on copyrighted software are now punishable under the DMCA. In addition, it is a civil violation and a crime to "manufacture, import, offer to the public, provide, or otherwise traffic in any technology, product, service, device, component, or part thereof" that "is primarily designed or produced for the purpose of circumventing a technological measure that effectively controls access to a work protected" under the Copyright Act.

A technological measure effectively controls access to a work if the measure "in the ordinary course of its operation, requires the application of information or a process or a treatment, with the authority of the copyright owner, to gain access to the work." 17 U.S.C. §1201(a)(3). One circumvents such a technology measure if one uses a means "to descramble a scrambled work, to decrypt an encrypted work, or otherwise to avoid, bypass, remove, deactivate, or impair a technological measure" without the authority of the copyright owner.

Patent Protection

Ideas, which are not protected by copyright, can be protected through a patent. In general, the patent laws protect the functionality of a product or process. A patent can be properly obtained if the invention is new, useful, nonobvious, and disclosed. The patent exchanges a grant of an exclusive monopoly for the invention in return for disclosure. Disclosure is the trigger point for patentability. The disclosure supports the claims of patentability; that is, it sets up the claim that the invention is both new and nonobvious and also establishes the scope of what can be protected. Thus, 35 U.S.C. § 112 states:

> The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same, and shall set forth the best mode contemplated by the inventor of carrying out his invention.
>
> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

A patent must disclose the best mode for implementing the invention, a clear written description of the invention, sufficient detail so that a practitioner can understand and make use of the description, and distinct claims in order for a patent to issue. By adequately disclosing the invention, the application gives notice of the technology involved in the patent so the public will know what would constitute an infringement. Because the application process can be expensive and because software is constantly changing, most software protection has relied upon copyright law.

The Copyleft: Open Source and Copyright

An open source license has several common features. It provides for (a) free redistribution;[2] (b) distribution in some form of the source code; (c) modifications, provided they are distributed under the same license; and (d) the integrity of authorship. Within the context of open source licenses, however, there are significant variations that can affect the future development and distribution of the product.[3]

The open source movement actually relies upon the same key features of copyright to protect a program as open source that proprietary copyright holders use to limit copying, distribution, and creation of derivative products. The copyright holder places on the program a notice that creates the license. The notice grants permission for a subsequent user to copy, modify, and distribute the program, but only upon compliance with the open source license, which must be either attached or incorporated into any subsequent distribution. Subsequent distributions also have to include the same notice, as do any works based on the program. Thus, the license reaches all works that are derivative works within the meaning of the copyright law.

An open source license prohibits restrictions being placed on the program that were not included in the original license. To put it another way, it requires the program and any derived works to be transferred with the same rights that the user received. It also requires that the distribution include or permit access to the source code.[4] Most importantly, for any modification, it requires that the modified files carry prominent notices displaying the date of the modification and the author of the modification. These notices create the integrity of authorship and provide a history of the program, and they cannot be removed. Each distribution of program must carry forward the notices of authorship with each iteration of the program.

[2]"Free" refers to the right to redistribute, not price. One can elect to distribute the software free or to charge for the service of distributing the software.

[3]See, for example, www.opensource.org for a listing of various approved open source licenses.

[4]Not all open source licenses require distribution of the source code. Some only require distribution of binary or object code for all or portions of the program. This is one of the key differentiating features among licenses.

Open source licenses also require that everyone understand that there is no warranty for the program and that the user receives the software "as is." All warranties and liabilities are disclaimed by the license under which the user receives the software. One may, however, sell warranty coverage, and businesses such as Red Hat have been developed around that model.

Finally, most open source licenses address the issue of software patents. The concern, of course, is that someone along the chain of distribution will claim a patent in the software and endanger the right to future distributions by claiming that some key feature of the program is proprietary. Thus, most open source licenses stipulate in one form or another that any patent that derives from the program must be licensed to all users free or is prohibited by the license.[5]

Failure to comply with an open source license will result in termination of the license and also constitutes a violation of copyright law.[6] Termination of the license, however, does not affect others in the chain who are in compliance with the terms of the license.

The various forms of open source licenses have not been truly tested in the courtroom. Such licenses are not usually signed, nor are they negotiated. Under traditional contract law interpretation, there may be an issue as to whether the licensee has manifested its assent to all terms of the license. While some in the legal community continue to debate the validity of these licenses, the growing view is that they are enforceable. The simplest analogy is to shrink wrap or click wrap licenses.[7] Much of the challenge to those licenses focused on the absence of actual agreement and the unevenness of bargaining power in the consumer setting. Nonetheless, the majority of courts have held these licenses to be binding from virtually any conduct that manifests assent after access to the license. In particular, the decision to install the software after reading the notices that are required to appear on startup will mostly likely be deemed sufficient manifestation of assent to enforce the license.

---

[5]The World Wide Web Consortium (www.w3c.org) is in the final stages of review for its royalty-free patent proposal, which aims to prevent blocking patents from interfering with critical components of the Internet infrastructure.

[6]It is an interesting question as to who will be able to enforce the license. Under the GNU GPL, the copyright ownership has to be transferred to the Free Software Foundation. But this is not true in other forms of the open source license.

[7]In the normal commercial context, assent to the terms of a license is shown by an actual signature of the party to be bound. Off-the-shelf software is commonly sold with a "shrink wrap" license, in which the opening of the package is the act that signifies the user's consent. With a click wrap license, it is clicking on "agreed" that shows assent to the terms of the license before software is installed but after it is purchased.

One Size Does Not Fit All

The two most commonly known forms of the open source license are the GNU General Public License, popularized by the Free Software Foundation, and the Berkeley Software Development license. Of the two, the GNU GPL, which was originally the less accepted of the two licenses, appears to have won out in the open source community.

GNU GPL

The GNU GPL requires full distribution of source code with any distribution of the original program or a derived work. In addition, the distribution must incorporate the text of the GNU GPL as the governing license. Under the GNU GPL, a developer may retain the copyright on his or her contributed code that is distributed as part of a software package. The developer can license that same code for others' use. However, if the code contains other GNU GPL contributions, then the code can only be distributed under the GNU GPL and cannot be distributed under a separate license other than by getting permission from all necessary copyright holders in the chain. While theoretically possible, obtaining such permission is practically difficult. Thus, code dependencies create a so-called "viral" effect as GNU GPL code is incorporated into other software products and compiled with them.

The GNU Library GPL (LGPL) is an effort to address some of these concerns. The LGPL was developed to permit proprietary software to link with GPL libraries. Essentially, it does not require that the GNU GPL be followed as long as the proprietary software is not compiled with the GNU GPL code. If the link is "dynamic" rather than "static," the two can be used in conjunction with each other under separate licenses. In the Linux context, for example, kernel-loadable modules that dynamically link to the already compiled Linux kernel have been treated as dynamic links covered by the LGPL. Thus, Linux is able to interface with many other proprietary devices and hardware drivers without "contaminating" the proprietary code.

BSD

The BSD license was used initially to build open source Unixes from the Berkeley Unix base. Unlike the GNU GPL, the BSD license permits but does not require the redistribution of the source code. If source code is redistributed, the distribution has to show copyright ownership and contain required licensing disclaimers. The BSD license also permits redistribution in binary form (object code) only so that software vendors can develop proprietary versions without redistributing the source code. Thus, for example, BSD code has found its way into Linux but also versions of Windows.

The BSD form of license was thought to encourage wide distribution without restrictions on use, including others selling proprietary programs that include BSD code. While this has led to wide distribution and incorporations of portions of BSD code in other programs, the problem with the BSD license is that it promotes forking. Competing proprietary versions can be, and have been, developed. One of the virtues of open source — development effort that can be unified into one project — is thus lost.

Netscape Public License

A third strategy is represented by the Netscape/Mozilla Public License.[8] In 1998, in response to Microsoft's Internet Browser, Netscape "open sourced" its Mozilla browser. In open sourcing the Mozilla browser, Netscape had to deal with embedded code from third-party developers that needed to be treated as proprietary and that shared code in the browser with Netscape's other proprietary server products. To address these issues, Netscape developed the Netscape Public License (NPL). The NPL permitted Netscape the time to reengineer its proprietary server products if it so chose. It also carved out for Netscape the right to continue to distribute the now open sourced code as part of its proprietary products.

Apart from these carve-outs for Netscape, the NPL distinguishes between modifications and new code. Modifications, bug fixes, and improvements to existing code must be distributed as open source with the source code for the modifications. New code, however, may be kept proprietary or not. Thus, for example, new libraries do not have to be open sourced. The NPL specifies that the code modifications needed to call the new library must be made available, although the code for the new library itself need not be.

The NPL attempts to balance proprietary and open source concerns. Because it permits portions of the distributed software to remain proprietary (i.e., distributed as binary code), the NPL also requires that the distribution be accompanied by a section entitled "Legal," which details the copyright and patent claims to the various portions of the distributed software.

[8]The Mozilla Public License (MPL) is essentially identical to the NPL except that it does not have the special privileges for Netscape.

IBM PL

IBM has followed Netscape's lead in coming out with its own public license for open source (IBM PL). Like the NPL, the IBM PL divides the code into contributor code (the original IBM-supplied code with contributed additions), which must be distributed as open source, and proprietary code, which must be segregated into separate modules and cannot be derivative. IBM handles issues of intellectual property by requiring any contributor to grant a license to all recipients to use the code in connection with the contributor code. Should any recipient bring a claim for infringement of intellectual property, that recipient automatically loses the license.

Current Legal Issues with Open Source Licensing

It should be apparent by now that open source licenses carry both benefits and burdens. Any company that is thinking of migrating over to open source platforms or using open source applications should carefully consider the implications of such use for protecting its proprietary systems and data.

Opponents of open source focus on the potentially viral or "contaminating" aspects of certain open source licenses to argue against implementing open source programs.[9] The myths and realities of these attacks are the subject of much debate. The open source community largely views such statements as scare tactics designed to discourage and limit the use of open source.

The argument that open source code will contaminate proprietary code, while it contains a kernel of truth, goes too far. There are any number of strategies that can be devised to retain proprietary programs as proprietary. Contrary to the impression being conveyed by open source opponents, a proprietary program does not become infected by coming too close to an open source program. There is no concept of "freedom simply by association." Nor need the proprietary status be lost simply because the programs "converse." Rather, one needs to examine carefully the open source license to determine how proprietary programs are addressed, if at all, and what makes a work a derivative work under the license. If compilation and static linking are the touchstones (as, for example, under the GNU GPL), then a different relationship needs to be devised. With a little legal help, IT personnel or consultants can find a way to avoid the problem.

[9]Indeed, one of the concerns regarding the adoption of UCITA is that proprietary developers will prohibit links of any kind with open source programs in order to stop the growth of open source and set the stage for a true software war.

Other attacks on open source have come through UCITA,[10] which has been adopted in Virginia and Maryland.[11] Because of concerns that UCITA would mandate certain provisions in licenses, the open source community has opposed its adoption. Recommendations have now been made to exclude open source from UCITA's operating provisions. Because of this exclusion, the provisions of UCITA that unequivocally recognize and convey legitimacy upon click wrap and shrink wrap licenses will not apply to open source.

Finally, one member of the Unix/Linux community itself has most recently launched an internal attack on Linux. The SCO Group announced during the recent *LinuxWorld Conference and Exposition* that it was investigating whether distributions of Linux were violating its intellectual property rights in Unix.[12] SCO claimed that companies using its Unix program had violated its nondisclosure agreements in developing Linux code or that users who had migrated from Unix were still using SCO Unix software components or libraries in violation of SCO's licenses. On 7 March 2003, the SCO Group backed up it threats with a lawsuit against IBM filed in Utah. According to SCO, the suit claims that IBM had, for the benefit of its new Linux services, misappropriated SCO's Unix code in breach of license agreements with AT&T, for which the SCO Group claims to be the legal successor.[13]

Microsoft has recognized the inherent danger open source poses to its position and has mounted a PR campaign promoting its "shared source" program. All versions of the shared source programs, however, forbid redistribution, sharing with third parties, or modifications of the code. One of the real disadvantages is that a user, at best, can make recommendations to Microsoft regarding bugs or needed modifications but has to wait for Microsoft to release a new version as it sees fit. Moreover, the shared source license requires that the code be treated as confidential proprietary data. This can lead to problems, such as when users develop adjuncts or parallel processes and need to prove that these internal developments were not infected by exposure to Microsoft's source code.

[10]See William Zucker, "Don't Fence Me In: UCITA, A Wakeup Call for Software Users," Cutter Consortium Business Technology Trends and Impacts *Executive Update*, Vol. 3, No. 8; and Cem Kaner, "UCITA Will Cause Short- and Long-Term Harm to the Industry and the Public," Cutter Consortium Business Technology Trends and Impacts *Council Opinion*, Vol. 3, No. 1.

[11]Maryland has adopted an amendment to UCITA to exclude the warranty of merchantability, implied by law under UCITA, from applying to open source licenses so as not to alter the disclaimer of any warranty in the license.

[12]The SCO Group's claims were not limited to Linux. It also claimed that Windows, Mac OS X, and versions of BSD infringed versions of Unix owned by SCO.

[13]For a fuller description of these claims, see www.olliancegroup.com.

The Decision to Open Source

Today, there are many significant open source software offerings available, including Linux, Apache, Perl, Sendmail, BIND, Samba, and MySQL. Open source has been adopted in the enterprise because of its ability to perform specific functions reliably, securely, and with dynamic scalability. It offers a less expensive competitive solution that is supported by leading systems vendors (e.g., IBM, Sun, Oracle, Dell, and HP) and by supporting services and warranties from major software vendors (e.g., Red Hat, Oracle, United Linux, Covalent, Sendmail, Inc., and Computer Associates). While there are legal issues, they are not insurmountable.

Because of open source software's lack of warranty or support, and given the way open source is developed, it is important to adopt open source technologies that have proven support services and are professionally documented. Organizations should take into account how open source will be integrated and whether particular licensing terms will affect their strategies for developing and marketing products and solutions. Although open source may not be appropriate in all circumstances, it clearly has won market acceptance and should be seriously considered by any organization looking to lower costs and increase flexibility.

CHAPTER 6

Open Source and the Cathedral

by Brian J. Dooley

The debate between the open source and commercial software communities continues to gather steam as the Linux operating system enters the enterprise in greater numbers. These two models are generally described as opposite in nature, with open source — so-called “free software” — presenting a direct threat against commercial software. In fact, some of the more radical backers of open source actually do see it that way; however, there are just as many backers of the opinion that open source is simply part of a larger picture of development and distribution that can actually encourage commercial software markets.

This debate has become increasingly important as open source software (OSS) makes further inroads into the enterprise, driven partly by the phenomenal success of Linux. The basic questions that must be answered include:

- Is open source ready for the enterprise?
- Can open source and commercial software coexist?
- What are the benefits of using OSS?
- What is available for the enterprise?
- How do the overall costs of OSS compare with those of commercial solutions?

In general, the backing of major vendors such as IBM have made open source credible to the business community over the past several years. The market dynamics, however, have been often misunderstood. This chapter will examine the open source phenomenon, and Linux in particular, and consider its place in the enterprise software environment.

The Open Source Debate

The commercial versus open source argument was really opened in Eric S. Raymond's seminal paper (and later book) "The Cathedral and the Bazaar" and his subsequent works, such as "The Magic Cauldron." The argument presented in these papers involves a fairly wide range of separate issues, each of which has progressed to some degree since 1998 when the first paper was published. The papers contrasted a centralized "cathedral" approach to a distributed "oriental bazaar" model and described a variety of components of the software environment using these models. Some of these areas are as follows:

- Development methodology: how should software projects be organized and assigned?
- Productization: should companies sell software or sell computer support that might or might not include software?
- Which is better: standardization in a distributed environment or hegemony in a monopolistic environment?
- How should software be distributed, sold, and priced?

Raymond, of course, favors the open source bazaar type of distributed development based on nonprofit incentives, but he does not argue that this should be the only model available. In fact, he calls for coexistence between the cathedral and the bazaar.

In the late 1990s, the debate generally focused upon this dichotomy of the software environment. However, although this model provides a nice explanation for many elements of software development and sales, it runs the risk of polarizing debate into two camps. In reality, the environment is much broader and requires careful consideration of what software represents as a commodity, as well as the numerous ways in which it has been sold, leased, licensed, given away, pirated, copied, and evolved.

It is only by looking at the whole development and sales picture that it becomes possible to understand the relationship between open source and commercial software. This leads to some answers to the questions of whether open source is ready for enterprise deployment and what role it is likely to play.

The Cathedral and the Bazaar

> Linus Torvalds' style of development — release early and often, delegate everything you can, be open to the point of promiscuity — came as a surprise. No quiet, reverent cathedral building here —

> rather, the Linux community seemed to resemble a great babbling bazaar of differing agendas and approaches (aptly symbolized by the Linux archive sites that would take submissions from anyone) out of which a coherent and stable system could seemingly emerge only by a succession of miracles. [13]

The Cathedral and the Bazaar (O'Reilly & Associates, 2000) has been raised as a manifesto for open software. This much discussed book, and supporting essays, has served to raise the profile of open software and present an intriguing picture of software development and sales as a contrast between a highly centralized model (the cathedral) and a highly distributed model (the bazaar):

- **Cathedral** — development is based on centralized planning techniques purported to be in use in traditional proprietary software development. In this model, development is organized top-down within a strong organizational hierarchy.
- **Bazaar** — development is based on decentralized planning and purports to explain OSS development. In this model, development is organized in a network. A community of developers works on parts of the software. Patches are brought together via the Internet on a central Web site.

Though there is much to be said for this view as a conceptual foundation, the truth is that the situation is considerably more complex and fluid. Software has a number of unique characteristics of development and distribution that make it fundamentally different from other types of merchandise. There are also some strong development and distribution differences between corporate, personal, and academic computing, and all of these factors feed into a highly complex equation. In a sense, there is no cathedral and no bazaar — there are only tendencies toward decentralization and toward centralization that appear in every phase of the environment.

The Development Model

First of all, software development has seldom achieved a truly centralized cathedral model. Large projects are built by large programming teams under centralized management. But they frequently include tools brought in or purchased from outside; reverse engineering of competing projects; free and open software tools and components developed in academia or for the government; and so forth. Large software pieces have also been more often licensed on a continuing basis, with updates and servicing included, rather than being sold as a commodity. This is true of almost all mainframe software, for example. In the academic environment, which arguably provides the true model of open software development, code has always been traded freely — very much in the manner of academic papers. To carry this analogy further, papers — like open software — are often compiled by profit-making companies and sold for a profit.

In fact, the free software movement might more readily be compared to cathedral building — bearing in mind that the Freemasons consider themselves the direct descendants of the cathedral builders. Historically, there was an overall plan overseen by a single architect or a committee (the program leader). Contributions were from an elite group of stonecutters — masons — who were skilled in their trade and possessed various secrets of the stonecutters guild. Rank and position within the guild would be determined by demonstrated experience, acclamation, and the passage of tests, resulting not in monetary benefit, but in increased prestige (ego points). Major contributions to the project had to pass muster with a committee (peer review), and there were often several alternate submissions to be evaluated — such as the famous doors on the Baptistry in Florence, Italy. As the cathedral belonged to the Church, neither the cathedral nor its appurtenances could be used for profit. In fact, the similarities to open software are quite interesting.

However, despite these factors, the point regarding centralization is duly noted. Building a cathedral today would tend to follow the highly centralized, rigorously hierarchical model. But this would be true of any large construction process: for example, with Department of Defense projects, such as the building of submarines, which is the original source of many of today's project management concepts. Such a model does, indeed, go against the free-flow development of open source. However, open source itself requires structure in order to exist. It requires project leaders, reviewers, and a "guild" of skilled craftspeople. It is important to note that the original image of the cathedral and the bazaar is actually two ends of a very complex continuum, which includes a wide variety of development, code sharing, and commercial models.

The Pricing Model

True commoditization only emerged in the software world with the advent of the PC and the development of "boxed" software that could be sold off the shelf. This created an illusion of a commodity software market. However, the PC also brought in a whole range of new free software sources. Piracy, for example, was (and still is) rampant. In fact, by providing "trialability," piracy helped vendors like Microsoft gain their success. Copies of office software were and are often brought home, creating "free copies" at home. Microsoft expressly permitted home copies at one time, though it has recently reversed this policy. In the early days, it may be noted, it was estimated that some three out of every four copies of the early word processing program WordStar were pirate copies.

Software is frequently bundled with new computers, creating an illusion of free software and insulating the purchaser from the listed price. In addition, there is an enormous body of freeware and free trial shareware available for the PC and Windows environment. So, at the distribution end, “free” software has played an important role in building the whole market — both commercial and open source. Vendors gained most of their revenues from sales of legitimate copies to businesses; from sales of software and support to those who made extensive use of the software and required support; through sales of service and support; and, importantly, through sales of upgrades.

In predictable fashion, prices for commoditized software became extremely high, relative to the cost of the hardware, because every commercial update requires new features. Software size and complexity rose rapidly. Low-end users and those who require, say, word processing at a sophisticated level only several times a month can hardly justify the high purchase costs. This results in both a tendency toward piracy and a tendency not to upgrade — until compatibility problems with newer versions of the program might emerge. Commercial vendors have fought back by strenuously combating piracy, providing “de-featured” versions of their software, and finally, by supporting outsourcing via an ASP model. But this only tends to confuse the potential purchaser as to the true value of the software — and the ASP model is really selling an access service rather than a commoditized product.

Open source and other software in this range are distinguished not only by factors of development but also by cost. Source code is provided, and the product is generally available in some form for free. But if software is free, where is the money supposed to come from?

Just where it always has, actually. Money in computing comes from service, installation, and support. It comes from customization, modification of software to unique circumstances, and from direct sales to organizations that can do with a prebuilt or pre-customized “boxed” solution. This box could be a product such as Microsoft Office — assembled by specially configuring and tweaking a number of products developed by different organizations. Or, it might be a pre-configured version of Linux, such as Mandrake.

Money, in other words, comes from service and convenience. The commodity market is, in fact, merely one aspect of this — a case where convenience and standardization are dominant.

Putting Them Both Together

The open software models that exist are likely to persist because they operate like the other mechanisms of the computing environment. Large projects, such as Linux, very much resemble a cross between a standards body and an academic conference. There is strong leadership, multiple submissions, and extensive peer review, and contributors gain recognition, experience, and "editorial review" — all of which can improve both position and skills. Great synergies can be obtained in this manner, and the success of the Internet standards can be seen as just as effective as Linux in pointing out this phenomenon. Multiple submissions and peer review can, indeed, improve product; it is also better to reuse code or start from an existing model than to build from scratch. But these factors have generally been known and always operated upon within all areas of the development community.

Free distribution is also likely to persist because it has always played a role in the software development and distribution environment. It is viable. However, it is not the only model. Variants of commercial distribution can and will continue to coexist. Large software systems, such as enterprise resource planning (ERP), customer relationship management (CRM), accounting systems, and the like tend to be sold under comprehensive licenses that include support. Specialty and niche market software may not attract the interest of the developer community and thus may be best provided as a commercial option. Very low-end utilities, games, and programs can be sold at low cost with little support as boxed products. Workstation and office products, however, fall somewhere in between. But today this area is sold on convenience and assured interoperability — factors favoring a single-vendor, easily installed boxed solution.

Software Distribution

There is not, and never has been, a single model for software distribution and sales. This is fundamental to any discussion of open source. Software began as tools that were developed inside companies and occasionally distributed, then grew into commercial products that were licensed for a term along with the hardware that supported them — with emphasis upon the hardware rather than the software. Licenses were tailored to the size of the hardware being installed. Then a variety of licensing and support models followed, with software becoming increasingly complex. But the PC brought a new range of boxed commodity products onto the market, which were licensed for an indefinite term and provided with only limited support. Meanwhile, throughout this period, universities, the Bell telephone monopoly, and the federal government were turning out innumerable software products that were released for free due to copyright requirements or by tradition. Development itself followed an equally wide range of models, from informal academic projects up to highly centralized US Department of Defense models in the public sector, and from single developers' somewhat chaotic efforts to advanced, centralized, hierarchical programming projects in the commercial sector.

The basic software models, characterized by distribution type, are as follows:

- Proprietary/commercial distribution, in binary form with no available source code. This has been extended to include cases where source code is included for reference only or where source code is included under the assumption that only the vendor or vendor-licensed operators will make changes. Sale is generally through licensing, which may be indefinite for smaller products or for a set time frame with conditions for renewal.
- Free, semi-free, and partly free distribution, much of which can be considered either open source or allied with open source.

Within the "free" camp, there are a number of important submodels, with the most important being those that claim to be open source. In addition to a distribution model, open source licenses have important consequences that encourage product development by an extended community of engineers. Free software variants include the following (not all are "free" in the sense of code sharing):

- **Shareware software** — typically provided in binary code only, at no cost (and often with some functional limits) for an initial period but requiring a purchased license after the passage of that period.
- **Freeware** — no license fee at all and generally released only as binary code. Freeware is often used as a "loss leader" to draw attention to a vendor's commercial products.
- **Open source software** — provided with source code and permission to modify. Remuneration differs. Most is available via download for free, but there are often configurations available for sale by vendors. The main characteristic is that the users can freely use, modify, and redistribute the software. A variety of licenses are available under this general category, some of which are:
  - **GNU Public License (GPL).** The most widely used open source license. Products under this license include the GNU (which stands for "GNU's Not Unix") project and Linux. The GPL is the most widely used OSS license. Key points of the GNU license include: software licensed under the GNU GPL can only be copied and distributed under this license; products licensed under the GPL may be sold; users can alter the source code, but if the result is distributed or published, it must be made available under the GPL license; ancillary technology can be developed, and as long as such products do not included code licensed under GPL, they need not themselves be licensed or made available under GPL; and the GPL represents the Free Software Foundation's (FSF) philosophy.

— **GNU Lesser General Public License (LGPL).** A less restrictive GPL variant developed by the FSF. It permits commercial software to use GPL libraries without requiring a standard and restrictive GPL license. The GNU C Libraries are released under this license.

— **Mozilla Public License (MPL), IBM Public License, and Sun Public License.** Beginning with the MPL for Netscape browser source code, these licenses impose some GPL-like restrictions on use of software, but like LGPL, licensed software can be incorporated into products that can be licensed without similar restrictions. These licenses are Open Source Initiative (OSI)–approved.

— **BSD and FreeBSD Licenses.** Grant the right to obtain a copy of the software including source code and documentation free of charge to everyone, as well as the right to trade the product commercially. Redistribution and use in source and binary forms is allowed.

Differences between the "free" models and the permissions they provide are given in Table 1 [4].

OSS and its variants are often also referred to as "free software." Free software refers to the user's freedom to run, copy, distribute, study, change, and improve the software, rather than to lack of cost.

Shared source is a term recently introduced by Microsoft to provide some companies with access to Microsoft source code for review purposes only. It does not, however, include the right to modify the code, so it is effectively meaningless.

Table 1 — Differences Between Free and Open Source Software Licenses [4]

Software license	Available at no cost	Distribution allowed	No usage restrictions	Source code freely available	Source code modification allowed	Derived work must be free	Linking with proprietary software allowed
Public Domain (Unrestricted)	X	X	X	X	X		X
Shareware	X[1]	X					
Freeware	X	X	X				
Open Source Licenses							
GPL	X	X	X	X	X	X	
LGPL	X	X	X	X	X	X	X
MPL	X	X	X	X	X	X	X
BSD/FreeBSD	X	X	X	X	X		X

[1]Shareware is free for a trial period only.

Open Source

Open source software has significantly different culture and economics than that of commercial software. Development occurs across a wide range of developers, many of whom are also users. For purists, payment is in ego points, but in a practical sense, most payment — if software engineering is the core occupation — arrives through service contracts, part of which may be obtained as business by demonstrated expertise in software creation.

Commercial software development tends to be monolithic. Management is hierarchical, and special management procedures must be put into place to ensure that maintenance is done and bugs are fixed, the optimum number of available programmers are used, and so forth. In open software, these issues are served by the community of interested users, augmented by the license that actually requires free distribution.

Maintenance has always been a problem for software. In the commercial (or factory) model, software is considered a commodity for sale, and funds are mainly derived from unit sales — with less money available from support contracts. Consequently, the emphasis is upon major releases that can provide funds, and support is handled with some indifference. Typically, new releases are treated as new products and require an array of flashy new features that are provided to disguise large sets of bug fixes. In open software, development is continuous, releases are frequent, and revenue is dependent upon service, so individual releases are of far less importance. Also, since service is constant, it provides revenues at a lower rate for a longer period of time. For commercial software, there will always come a time when all the required features are in place, and in times of economic downturn, new versions and updates may simply not be undertaken. It is partly for this reason that Microsoft is initiating a more draconian licensing policy, which virtually requires constant updates under the threat of vastly higher prices when new versions are required.

A number of studies have been done through the years on the relative costs of running open source and commercial products. The results have generally been ambiguous, heavily dependent upon the particular situation under consideration — and who paid for the survey. In one infamous case, a comparison showed Windows as having an advantage over Linux; the survey was later shown to have been paid for by Microsoft, with its equipment carefully optimized for the situation against a Linux system that was not. Culture plays a large role in this debate — not only within the Microsoft culture, but also within Unix, which has always operated as a closed and highly technical community with a particular devotion to its operating system.

Claimed Open Source Advantages

Open source proponents, using arguments based on essays such as "The Cathedral and the Bazaar," argue that open source software has the following strengths as compared to conventional commercial software:

- **Security.** OSS is always peer reviewed, and problems are quickly identified and fixed. Commercial software relies on the fact that hackers can't see "under the hood," but this results in less peer review and fewer people to identify and patch bugs as they occur.
- **Stability.** Open source peer review, in theory, promotes stability by locating and fixing flaws very quickly or forking the product to new "ownership" if stability cannot be ensured. Commercial software is designed to go out the door as soon as possible in "just good enough" condition, which can mean that the product is not fully tested and, hence, not fully stable.
- **Flexibility.** Open source is designed to leverage existing material, and any problem will have dozens of people working on a solution. Commercial software flexibility is limited and costly.
- **Scalability.** Lack of licensing requirements in open source increases the possibility of scalability. With commercial software, licensing can get confusing and expensive, reducing scalability possibility.
- **Longevity.** Open source programs run on numerous devices, and the community continues to exist even if the developer goes away. Commercial firms may abandon products or lose ability to patch them as key programmers move away.
- **Interoperability.** Open source is built and based on standards, with most programs interoperating at a basic level since inception. Interoperability has often been a problem for commercial systems due to proprietary interests.
- **Ease of administration.** Powerful administration features are provided in Linux and other open software products, but they do require considerable knowledge to be correctly administered. Commercial products require less knowledge, but this too can be a trap since the programs are inherently complex.

These claims are not always met, but they have been put forward partly to counter the misgivings of users who might be reluctant to rely on anything that sounds like it might be free. But open source is not without its negative characteristics, which, like the positive, are not universal but characterize some portion of the products available. These negative aspects include:

- **Relatively few features.** As a base example, all of Linux has about 1.5 million lines of code, while Windows has more than 30 million. Some feature sets are rich — Star Office, for example — but many of these cases are strategic releases designed specifically to thwart Microsoft or provide a program substitute.
- **Less graphic sophistication.** Linux still has several graphics environments — KDE (K Desktop Environment), GNOME (GNU's Object Model Environment), plus several others — but graphics are relatively new on Linux, and sophisticated graphics are difficult to develop. This partly explains why relatively few high-end games are available.
- **Clumsy installation.** Even with standardization on Red Hat's RPM packages, installation of software can be problematic, and troubleshooting new installations can be difficult. The file system is cumbersome, and locations of items can be difficult to find.
- **Too much, too many.** Linux software presents a sea of possibilities, and selection is often difficult. Even on a system set up from a standard distribution, there are often numerous overlapping configuration utilities, unexplained programs, and programs that are not named in intuitive fashion (no doubt, a continuation of a tradition to name utilities with names such as GREP, NROFF, and so forth).
- **Support for some, but not all, expected functions.** This is a distinction that is beginning to disappear, particularly as the battle heats up (relatively) over workstation space.
- **Lack of communication between products.** This is true of using products from diverse manufacturers of any type on any operating system. Standards help here, but integration between programs by the same vendor is always tighter than integration of products between multiple vendors. This, too, is improving with the movement of Linux down to the workstation.
- **Fragmentation of packages and installation routines among diverse operating system distributions.** This is starting to clear up with Red Hat's commanding market share and the development of UnitedLinux, but it still persists.

It is important to note that open source does not refer to the Linux environment alone. There is a considerable product base of open source products available now for Windows as well as for proprietary Unix. Although such distributions tend to go against the original "purist" spirit of the open source movement, they are, perhaps, inevitable. Only a few, such as the ubiquitous Apache server, have become common under Windows.

Open Source in the Enterprise

OSS has largely entered the enterprise through the back door. Much like early PCs, OSS has been confined in the past to technical areas, scientific workstations, and special applications such as intranet Web servers. Recently however, with growing support from major vendors and a significant presence developing in key areas — combined with significant publicity — OSS has reached a level of acceptance in the enterprise.

What Are the Open Source Enterprise Targets?

Most of the current opportunities for OSS lie in the server and mid-range application area. This is the territory currently served by traditional proprietary Unix systems. Microsoft is making a strong showing as well through its Windows 2000 rollout and advanced communications utilities.

A 1999 Datapro survey shows that Linux, leading the open software advance, is used most often for Web servers (33%), followed by scientific/technical applications (15%), with lower figures for application servers (10%), enterprise systems (10%), networked workstations (10%), and finally desktop applications (6%) [15]. Although these figures date back to 1999, they have remained relatively constant, and they reflect general tendencies in the overall Unix marketplace, only slightly affected by Linux's lower cost. They are borne out by a 1999 IDC survey across the board of Linux, Unix, and Windows NT, demonstrating where most spending went for each operating system (see Figure 1) [2].

The most prominent market is the workstation market, and it is here that hearts and minds are won — as well as market share and proprietary lock-in. Microsoft retains strong control over this area, but licensing schemes tend to make it expensive. Although in previous years, significant advances occurred with virtually every new version of MS Office, the rate of innovation has slowed to the extent that the newest features are of lesser interest. At the same time, Microsoft's new licensing schemes are adding to cost.

Market development has depended upon a three-year replacement cycle, enforced by incompatibility between files created by newer versions and those created by previous versions. Each new purchase of systems brings in bundled copies of the latest software to ensure that the process continues. The upgrade price is high and can involve many thousands of workstations. Even with special deals, this remains extremely expensive.

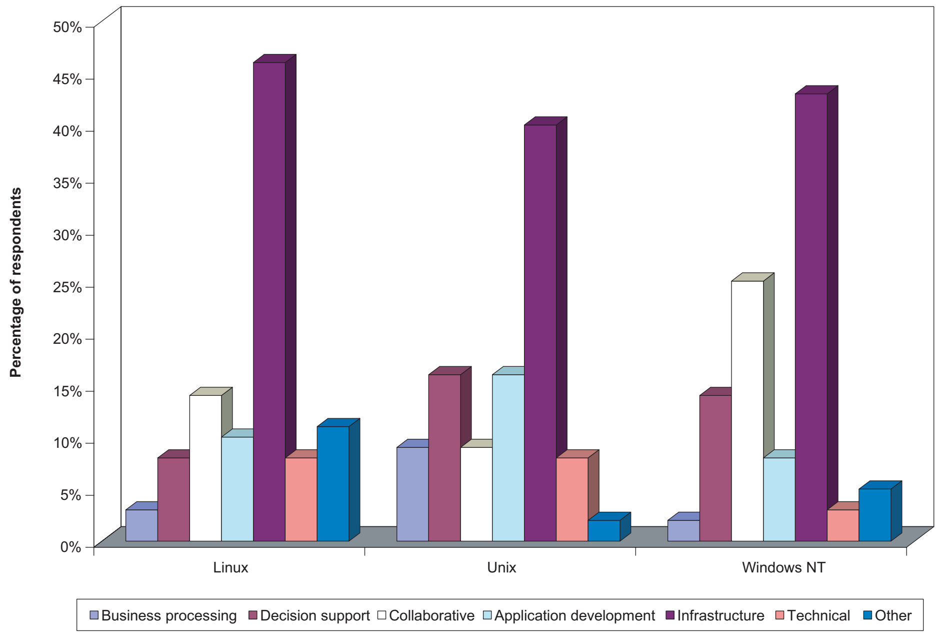


Figure 1 — US server workload by operating system [2].

Nevertheless, although companies are now somewhat more willing to seek alternatives to avoid the update cycle, this has not had a significant effect so far. The recent move of Linux distributors into development of special workstation packages demonstrates the growing interest in this area.

Operating Systems

There are two basic operating systems within the open source environment: Linux and BSD. Both of these systems have strong followings, though Linux has since surpassed all early growth expectations and captured the lion's share of media attention — and developer support. The fact that they are both based on Unix derives from the original licensing situation in which Unix — developed at Bell Labs — was originally released. As Unix developed, it grew into a variety of strong commercial systems as well as free versions under various licenses, which evolved from the peculiarities of the original licensing situation.

The commercial variants remain and present top-end competition to the open source versions. The major versions — IBM AIX, HP-UX, and Sun Solaris — are mature and well proven, known for reliability and scalability. They each have a strong and loyal user base. However, license costs remain high, with a Trusted Solaris 8 media kit costing US $2,600 for the 1 to 2 CPU version. They have long powered high-end systems and have proven themselves in an enterprise context. In fact, it is the presence of these systems that is gradually drawing "free" Unix further into the enterprise.

The Value of Linux

Linux gains its name from its operating system kernel, which was developed and is fully controlled by Linus Torvalds. Linux is deployed with the GNU tools to provide a complete operating system. It is an open source product supported by a community of thousands of developers and released under the GPL. The operating system has grown in importance, mainly in the server market and partly as a substitute for commercial low-end Unix systems — or as a replacement for Windows NT/2000.

In 1997, OSI was founded to solidify free or open source licensing. OSI was based on the "Debian Free Software Guidelines" (published in 1995) with input from Eric Raymond and Bruce Perens. OSI developed the Open Source Definition (OSD), which provides a guideline and trademark for non-GPL OSS software licenses. OSD-approved licenses operate according to the general model of the LGPL. To assist in further acceptance of OSS in the corporate world, the term open source software — instead of free software — was established.

Linux is a solid operating system, highly scalable, and known for reliability. It has grown up from its original character/command-based interface to provide a full GUI environment — with several possibilities. It has a strong community of users, and there is plenty of software available. Although statistics are unreliable on this point, it likely powers the majority of servers on the Internet through the Apache server program. It can be obtained for free, and even the priced versions are available at considerably less cost than commercial alternatives.

Linux is related to Unix and is gradually taking over for commercial Unix systems. It is actually a combination of an operating system written by Torvalds and the GNU Unix-alike tool set. Unix applications can generally be ported to Linux with relative ease because of the high degree of similarity that remains between commercial Unix and Linux. The trend is now toward development for Linux, particularly at low- to mid-enterprise level, with support from major hardware vendors, including IBM.

Linux is built, marketed, and distributed according to the open software or free software model, in which source code must be distributed for free and end user code improvement is actually encouraged. It is widely available over the Internet and on CD in a large number of distributions, each providing the core Linux components in their latest approved distribution, plus accessory products — generally for setup and configuration — and product sets designed for different environments. All components distributed, including distributor specialty software, must also be made open source (free), and the distributor makes money by providing a subset or superset of Linux designed for particular sets of users as well as providing service and support. Service and support contracts are critical in the enterprise environment, which is why some distributors, such as Red Hat, continue to do well even though the software is, in theory, free.

Among major Linux distributions, Red Hat remains king with well over 50% of all Linux systems. Other large players include Debian, SuSE (popular in Europe), Slackware Linux, and Mandrake (designed for beginners). The battle with Microsoft has resulted in a new distribution aimed specifically at the Windows workstation market, called Lindows. Lindows is attempting to differentiate itself from the overall Linux environment by promoting user-friendliness and availability of office applications.

Linux has a reputation for being extremely difficult to set up and configure, and it requires extensive support. This has always proven to be a major thwarting factor in the proliferation of all Unix-based systems. It has been used successfully against Unix and then Linux by every small-system operating system that has emerged. Some of these issues are being addressed, but much of this is very recent.

Linux has shown a recent growth spurt, partly due to policies regarding licensing by Microsoft combined with moves by major Linux distributions to develop products tailored specifically to the end-user desktop. All of the major distributions will have workstation distributions within the next few years, including newcomer Lindows, which has selected this as a niche market.

There are major differences in the structure of various IT markets that influence the position of open software (particularly Linux) versus conventional commercial software. In the server area, commercial software has less of a hold, particularly in the area of Web servers where Linux may actually be dominant. Linux is also popular in technical and scientific environments, just as Mac OS is popular in graphics. Most other areas are dominated by commercial software, with the important desktop area being dominated by Microsoft.

Microsoft's stronghold on the office suite market, unifying its operating system and its Office products, makes it difficult for other operating systems to compete. Since everyone needs an office suite, it often seems to make sense to put all the tools on Windows and take advantage of natural interoperability among Microsoft products rather than using several operating systems — perhaps on a single dual-boot PC — and attempting to develop connections between them. This is what makes Microsoft Office a fundamental part of

Microsoft's sales policy; where Office is in place and used by most of an organization, it becomes difficult to shoehorn another operating system in.

The BSDs

FreeBSD is known for stability, security, and performance, but it has been unable to draw on the great variety of both commercial and open source applications that are available for Linux. BSD (Berkeley Software Distribution) began as a Unix operating system toolkit, widely distributed as an enhancement to Bell Labs' Unix. As the toolkit evolved, it gradually replaced all of the missing parts of the operating system with files created and distributed under its own licenses. Although FreeBSD is the most well known, there is actually a family of BSDs, which includes the following:

- **FreeBSD** — basic BSD; the best-known variant
- **NetBSD** — a highly portable BSD, similar to OpenBSD (a spinoff)
- **OpenBSD** — a highly secure version of BSD, requiring significant technical skill
- **Darwin** — foundation of NeXT's NeXTStep and now Apple's Mac OS X

BSDs have been favored over Linux for installations with special requirements for robustness and stability.

Much of the BSD code was developed in taxpayer-supported research and so is free of charge. In fact, most operating systems today have some BSD code within, including Windows, OS/2, Linux, and Unix. The Internet's TCP/IP stack, in fact, comes from BSD, where it was incorporated into the kernel at the Defense Advanced Research Project Agency. The BSD license is truly free, meaning that portions can be altered or sold at will, without the "copyleft"[1] provisions of the GPL (except for Darwin, which is controlled by Apple and the Apple Public Source License).

One problem with BSD is that Berkeley is no longer producing new releases, and BSD is now defined by the separate development groups.

FreeBSD, now at version 5.0, has the largest development team, user base, available applications, and general activity. It runs on several processors, but Intel remains primary.

The BSDs are complete and highly integrated operating systems, with an internal consistency that Linux cannot (yet) match. Development also tends to be extremely conservative and relatively slow. The three BSDs create some fragmentation, but there has been increasing convergence. They will never merge, however, as they are designed for different purposes.

[1]See www.gnu.org/copyleft/copyleft.html for more information.

Although FreeBSD lacks Linux's presence and headlines, nearly 7,000 applications are available for end users and programmers. The latest release, FreeBSD 5.0, shows the result of continuous and steady improvement, particularly in security, hardware support, and developer tools.

Most of the applications available on BSD resemble or are modified versions of programs also available on Linux. Among these are OpenOffice, GNOME, and KDE. Installation remains character-based, and the system is even less user-friendly than Linux, but it has a solid following in its market niche.

Linux in a Microsoft Environment

There are several ways in which Microsoft Windows and Microsoft Office interface with Linux, such as:

- **Running Office on Linux.** This is possible through a variety of methods, including the Wine Windows substitute program that permits Windows programs to run under Linux (but not well). There are also a variety of systems that make it possible to run a copy of Windows under Linux and run Windows applications under that copy. Thirdly, a dual-boot system can be set up, in which the user can select either a Linux environment or a Windows environment at boot time.
- **Running a compatible application.** This is a better method and is now endorsed by Lindows (originally a Wine supporter). There are now a variety of Linux programs that support tasks provided by Office and other commonly used Windows environment products. StarOffice, from Sun Microsystems, is often quoted as a prime example. It provides word processor, spreadsheet, graphics, and presentation tools in a single integrated system that can read and create files compatible with the Microsoft equivalent.
- **Networking with Microsoft.** Linux has evolved a number of solutions for networking within Windows networks. The primary tool, called Samba, achieves this by emulating a Windows NT server. Although Samba can be somewhat difficult to configure, it does provide a highly robust solution to the networking problem.

Other standards originally designed for interoperability between other operating systems also ease the process of integration. For database products, as an example, SQL has emerged as a strong interface standard that can be used to link clients and databases across Linux and Microsoft, as well as with databases on other operating systems.

Significant Open Source Products

Among the myriad OSS products, the following are of particular note for a history of enterprise use. (Linux, perhaps the most important example, is discussed previously in this chapter, as is Unix-alike competitor BSD.)

Apache

Apache is based on the National Center for Supercomputing Applications (NCSA) Web server. The Apache Software Foundation, founded in 1999, is now responsible for its development. The Apache Web server has dominated its market since 1996. It is included in many software solutions, including IBM's WebSphere.

Internet Infrastructure Components, including BIND and DNS

The Berkeley Internet Name Domain (BIND), delivering the domain name system (DNS), turns host names into IP addresses — a very important component of the Internet. BIND is included in all Unix systems, as well as in many others.

GIMP

The GNU Image Manipulation Program (GIMP) is an OSS graphics program offering capabilities similar to Adobe Photoshop. It was developed initially for Unix systems and is now part of most Linux distributions.

GNOME

GNOME competes with KDE as an open source desktop environment for Unix, Linux, and other Unix-based operating systems. The GNOME project is supported by Sun, IBM, and Hewlett-Packard (HP) and has a strong development and user community.

GNU

Richard Stallman started GNU at MIT in 1984 in an attempt to build a free Unix-like operating system. The GNU project has not yet succeeded at building its own OS kernel (known as Hurd); however, when combined with the Linux kernel, it creates the successful operating system known today as Linux. The GNU project today is involved in many software projects related to GNU Unix and Linux. Among these are the GNU C Compiler (GCC — the de facto standard compiler for Linux programs written in C and C++), the GNU Privacy Guard (GnuPG), GNU Emacs (a text editor), the GNOME desktop environment, and numerous others.

KDE

KDE provides an alternative to GNOME. The project began in 1996 as an effort to develop a GUI for Unix. KDE is released as GPL compatible, and its libraries are available under LGPL, easing commercial software development.

KOffice

KOffice is an open source office application designed to work with KDE. It provides most of the functionality of suites such as Microsoft Office.

Mozilla

Mozilla is Netscape's open source browser project. When Microsoft began offering Internet Explorer for free, Netscape released its code under the Mozilla Public License. The 1.0 version was released only in 2002.

MySQL

MySQL is a relational database server, developed initially in 1994 by the Swedish company TcX DataKonsulter AB. The software was published under the GPL (some parts under the LGPL) in 2000. Licensed versions are also available that enable the owner to use MySQL in commercial solutions. MySQL AB owns the copyright.

Perl

Larry Wall developed the Perl scripting language in 1987. Perl then evolved into a network and systems administration tool. Its CGI-programming functionality made it the "glue" of the Internet, especially suitable for dynamic Web pages.

PostgreSQL

PostgreSQL is an object-relational database server project that was started at Berkeley. In 1996, a team there developed it into an open source SQL database. It is mainly deployed in private and non-mission-critical business operations. Red Hat offers a database product built on PostgreSQL 7.1.

Python

Guido van Rossum released the Python scripting language in 1991. It evolved quickly into a powerful object-oriented, interpreted programming language. The product is offered in a version completely integrated with Java (JPython), enabling it to run on every computer with the Java Virtual Machine.

Samba

Samba is a Windows file server and print server for Unix platforms. It was developed in 1993 by Andrew Tridgell of the Australian National University, who still leads the project. Samba is included in most Linux distributions. Samba has proven a robust connection architecture for Linux/Windows environments, actually performing its tasks more efficiently and faster than Windows.

Sendmail

The Sendmail program handles the majority of mail traffic on the Internet. It was developed in 1981 by Eric Allman at Berkeley as a mail transfer agent (MTA). Its focus is on openness in handling differing mail protocols, on routing functionality, and on flexible configuration.

StarOffice/OpenOffice.org

The StarOffice suite is a Sun product competing in the office suite market dominated by Microsoft Office. Sun made the StarOffice code available as the OpenOffice.org project. It can now be used by everyone, with competitive advantage gained by adding proprietary extensions. Sun's StarOffice product, for example, includes a number of extensions including a proprietary spell checker, which since May 2002, must be licensed with a fee.

Tcl/Tk

Tcl/Tk is a scripting language comparable to Perl and Python. It was developed by John Ousterhout at Berkeley and designed as a toolkit for developing GUIs.

Zope

Zope is an open source application server based on software that was originally proprietary. Rob Page and Paul Everitt developed the project in 1995 as the free open source toolkit Bobo and the commercial Web application platform Principia. Today, Zope is known as a competitive alternative in the application server market, particularly for content management and portal applications.

GNU Enterprise

The general role of open source software for the enterprise includes numerous programs running on a more-or-less hidden basis in the technical departments and deep within IT departmental operations. One intriguing open source project designed for the main market is GNU Enterprise (GNUe).

GNUe is intended to provide a suite of tools and applications for solving the needs of the enterprise, from human resources, accounting, CRM, and project management to supply chain management or e-commerce. Proposed components include:

- Financials, including:
  - Accounts payable
  - Accounts receivable
  - Cash management
  - Fixed assets
  - General ledger
  - Allocations and recurring transactions
  - Budgeting costing
  - Financial statements
  - Activity-based costing
  - Bank reconciliation
  - Investment management
  - Project-based costing
  - Consolidated financial reporting
- Customer relations
- Forecasting
- Human resources, including:
  - Personnel
  - Payroll
  - Recruitment
  - Skills inventory
  - Compensation management
  - Organizational planning
  - Compliance
  - Training
  - Benefits
  - Time and expenses

- Manufacturing, including:
  - — Bill of materials
  - — Capacity requirements planning
  - — Engineering
  - — Master production scheduling
  - — Material requirements planning
  - — Work in process
  - — Repetitive manufacturing
  - — Process manufacturing
- Service
- Quality
- Project management
- Sales
- Supply chain, including:
  - — Order entry
  - — Inventory
  - — Manufacturing
  - — Shipping
  - — Purchasing
  - — Billing

In general, most of the tools are usable and are actually in use at a number of live sites. Most of the actual packages, however, are at proposal stage or earlier, with no likelihood that any version will be seen soon. GNU Enterprise began as GNU G/L as early as 1996 and was revived and regeared in March 2000.

Linux

Linux is the flagship of the open source movement, and its success has played a large role in providing an open source foothold within the enterprise. It is also significant that the Linux distribution community has responded to the needs of enterprise users by developing more stable and robust versions designed for larger systems and in developing more user-friendly versions for use on the desktop. Development has been further spurred by major support from IBM, including special extensions allowing it to operate on S/390 mainframes, as well as provisions for multiple virtual Linux sessions under IBM AIX.

The Linux Juggernaut

According to a study by MITRE, more than 120,000 programmers contribute to Linux, volunteering about US $2 billion worth of labor [9]. There are about 204 unique Linux distributions on the market. Vendors ease the configuration and integration process by providing their own utilities (which must also be released under the GPL) and package configuration. They also provide sales, support, and integration, emphasizing service over product.

A recent study by The Goldman Sachs Group found that Linux has “an established and growing presence” in the enterprise market, with 39% of a sample base of 1,000 IT managers at large US-based multinational companies deploying Linux in some capacity [8].

Calculating an exact market share for Linux is difficult because there are installations from anonymous FTPs, multiple installations from a single set of commercial disks, and second-tier vendors providing CDs of downloaded copies of Linux distributions. The number of users has been variously estimated as 4-27 million in 2000, and most available evidence shows that it continues to grow rapidly, although market share remains insignificant relative to Windows and in the workstation market. According to IDC, Linux has gained market share every year since 1995 but still only controls about 2% of client operating system shipments [5].

The Linux Move into the Enterprise and Workstation Markets

Linux in the Enterprise

In 2001 and 2002, Linux distributors, including Red Hat and SuSE, developed special “enterprise” versions of the software, which included a support model similar to that of traditional enterprise-level products, with fewer releases.
To compete with individual enterprise releases, a number of Linux distributors came together to create a similar product, called UnitedLinux, that would compete principally against Red Hat Advanced Server. The initiating players are well-known Linux distributors Conectiva SA, SCO, SuSE, and Turbolinux, Inc. UnitedLinux has received substantial industry support, though companies expressing support were not required to actually contribute or put policies in place — so this must be considered more as a gesture than as an offer to promote the architecture. Supporting companies and organizations are shown in Table 2.

Table 2 — Companies Supporting UnitedLinux

4Front Technologies	Fujitsu	Network Appliance
AMD	Fujitsu Siemens Computers	OpenForum Europe
Arkeia	Hewlettt-Packard	Parasoft
BakBone	IBM	PolyServe
Basis International	IConexio Technologies, Inc.	Progress Software
BEA	Linux International	Toshiba
Borland	Microlite	Ximian
Computer Associates	NEC	Yosemite Technologies

The advent of enterprise Linux versions has led to greater acceptance at this level, a trend that has been bolstered by release of standard commercial software on Linux by major vendors.

Conventional software products available for Linux cloud the open source issue somewhat. These include Oracle, HP, IBM (DB2, VisualAge, WebSphere MQ, TX Series), Lotus (Domino), Tivoli, Transarc, Computer Associates (Unicenter), Sybase, Informix, SAP WebTrends, Netscape, and Sun. Versions of the major ERP programs are also available.

IBM has heavily endorsed Linux and open source products. One key release is IBM's Eclipse, an open source integrated development environment (IDE) that competes with Microsoft Visual Studio .NET. It also competes with Sun's open source IDE, Sun ONE, which is based on NetBeans.

Database solutions constitute a major portion of enterprise infrastructure requirements. Oracle has included Linux support since 1998. IBM's DB2 provides a control center for maintaining databases that can be run from a Java-enabled browser. MySQL provides a fast and robust relational database management system, which can be run on Linux and can connect to Microsoft Access running on a client PC. PostgreSQL is a sophisticated object-relational database system that is available on Linux.

The role of Linux within the enterprise generally corresponds to the low end of the server market, where lower scalability than Unix and greater perceived reliability and cost advantages over Windows give it an advantage. Situations in which Linux can excel include:

- Software that runs well in a distributed environment
- Compute-intensive application sets requiring high performance
- Applications currently running under Unix, but underutilizing system performance
- Applications whose demands on server resources are growing

- Software developed inhouse or which has been ported to Linux for at least one year
- Web servers, file servers, firewalls, application servers, and distributed application environments
- Production environments for testing and integration

Examples of specific situations that favor a Linux solution include:

- File/print services for Windows domains, using Samba
- Mail consolidation with Bynari Insight or Sendmail
- Web server and Web applications with Apache and commercial components
- Network service consolidation with open source tools
- SAP application servers
- Data warehouse and data mart solutions
- Small database consolidation including DB2 and Oracle
- Education applications

It is estimated that more than 36% of all Web servers run on Linux, though there may well be more due to the hidden nature of Linux distribution. Linux is now the fastest-growing operating system, with a growth rate of about 30% per annum.

D.H. Brown Associates has concluded that “the leading Linux distributions are now quite capable of serving as general-purpose operating systems for a broad range of departmental and workgroup applications” [1].

MITRE created an extensive report on open source software in the enterprise and came to the following conclusion:

> OSS is a viable long-term solution that merits careful consideration because of the potential for significant cost, reliability, and support advantages. However, these potential benefits must also be carefully balanced with a number of risks associated with OSS approaches and products. The optimal choice of OSS versus traditional COTS varies according to the specific requirements and runtime environment of the software. OSS is often a good option for products relevant and interesting to a large community with highly skilled developers. It typically compares favorably for server and embedded system implementations that may require some customization, but fares no better than traditional COTS for typical desktop applications. [9]

Analysts have estimated the total worth of the Linux market at about $4.6 billion in 2002, possibly rising to $7 billion by the end of 2003, with half of this coming from adoption in the enterprise.

There remains considerable resistance to Linux in the enterprise, however. Some of these factors are provided in a survey conducted by *Information Week* in 2000 (see Figure 2). Although the new enterprise versions of Linux are designed to address some of these problems, it may still be some time before they are reduced significantly.

Linux Applications for the Enterprise

There is an enormous and growing body of software available for Linux, including both open source and commercial applications. Development of enterprise solutions has experienced rapid growth recently, with many developers coming on board to support IBM's new zSeries — a top-end system designed to run multiple "virtual" Linux sessions in an enterprise environment. As of now, 135 software developers have listed 255 applications in this category with IBM.

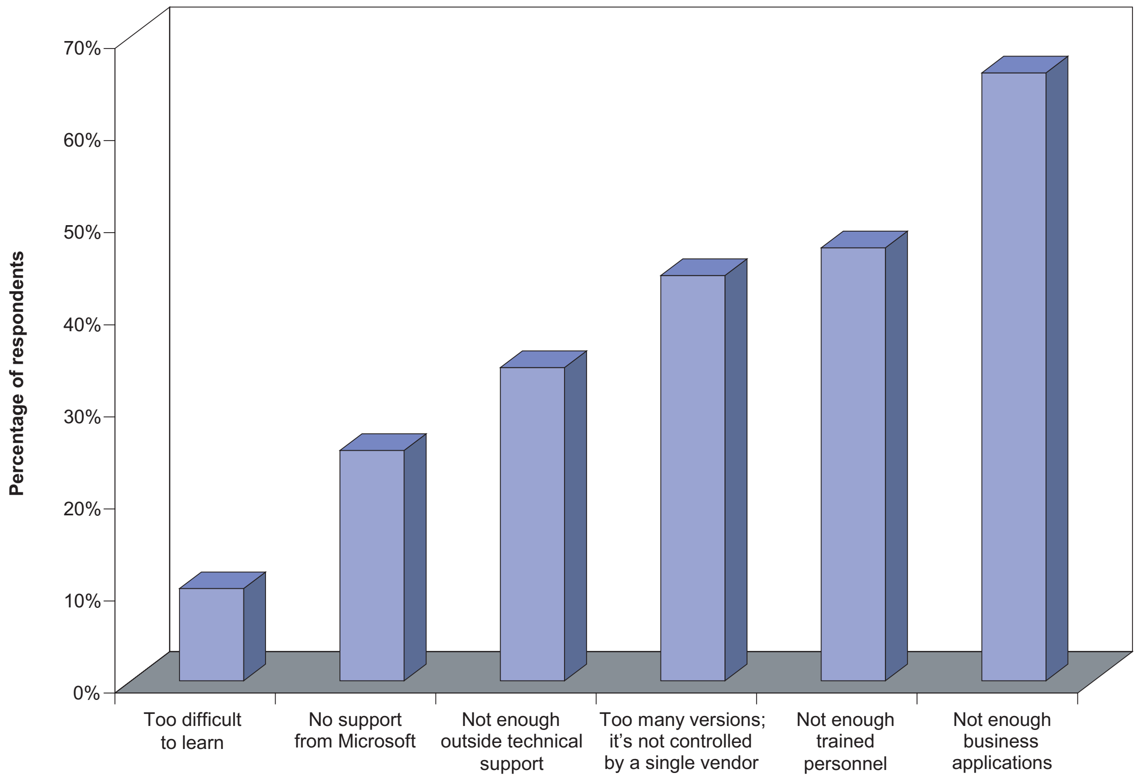


Figure 2 — Most significant weaknesses of Linux. (Source: *InformationWeek*.)

Linux and Open Source on the Desktop

Linux on the desktop has been more problematic, for a variety of reasons:

- Linux is perceived — and is — more difficult to use than Microsoft products, particularly in the critical areas of setup and installation.
- Adequate software capable of challenging Microsoft's position has only recently become available.
- Microsoft Office has an entrenched following and uses its monopoly power and licensing agreements to block entrants of competing systems.

It should be noted that open source applications are also available for the Windows environment but show no current strength, because Microsoft is able to use its marketing machine effectively for products running under its operating system.

Nonetheless, there has been an interest in workstation versions of Linux products. Red Hat has announced release of a Linux workstation version to be available in the first quarter of 2003. SuSE Linux AG is also expected to release a workstation version. Sun is moving into this sector as well, releasing a branded version of Linux, targeting the workstation and low-end server markets.

For workstations, the programs of importance are office suites and related applications, an area dominated by Microsoft Office. There are two methods being used to coexist with Office: to run open source solutions that provide file compatibility and similar functionality, or to run emulators and specialty products that make it possible to run Office on Linux.

Substitute programs have developed significantly over the past several years, although versions have always existed. The programs feature similar operations and similar performance and often generate compatible files. Some of these substitutes are listed in Table 3.

MS Emulators and Coexistence

Windows applications under Linux:

- **Win4Lin** — installs an emulator that runs Windows
- **VMWare** — installs a PC emulator onto which any PC OS can be loaded, including Windows
- **Bochs** — installs a PC emulator, like VMWare
- **CodeWeavers CrossOver Office** — permits Office to be loaded on a Linux PC

Table 3 — Linux/Open Source Substitutions

Commercial Program	Linux OSS Equivalent
Microsoft Office	StarOffice 6.0, OpenOffice
Microsoft Word	StarOffice Writer, OpenOffice, Write Pro
Microsoft Excel	StarOffice Calc, OpenOffice, Spreadsheet Pro
Microsoft PowerPoint	StarOffice Impress, OpenOffice, Presenter Pro
Microsoft Outlook	Netscape Mail, Evolution
Microsoft Internet Explorer	Netscape Browser, Mozilla Browser
Microsoft Project Manager	MrProject
Adobe Photoshop	GIMP, Photogenics
Adobe Illustrator	Sketch, Illustrate
Microsoft Project	MrProject
Microsoft FrontPage	HTML Editor, BlueFish
Microsoft Visio	Chart Pro, Flowchart Pro
AOL Instant Messenger	Netscape AIM, AOL AIM, GAIM
Real Player	Real Player
CD Burning	K3B CD Burner
Time Management	Time Organizer, Pilot Sync
Adobe Acrobat Reader	Adobe Acrobat Reader
Music Digitizing	G-Rip

- **VNC (developed by AT&T) —** permits remote control of a desktop under Linux, including remote control through a browser
- **Wine** — installs a Windows API

The Wine project is particularly significant, as an attempt to implement an open source Windows 95 API capable of running under Linux. So far it has not quite lived up to the promise of easy and seamless operation, but work continues. The list of applications with which it works is relatively slow, and there are other problems as well, such as slower operation and unexplained bugs.

Linux Distributions

Red Hat owns 60% of the Linux market. Other distributors are creating alliances, including those in the UnitedLinux effort designed to develop an enterprise Linux version in competition with Red Hat's Advanced Server.

Some of the most common Linux distributions, as well as their pros and cons, include:

- **SuSE Linux (8.1)** — has a reputation as one of the most comprehensive, stable, and secure operating Linux versions available; provides excellent documentation, with good included software selection and disk partitioning utilities; is not as strong in the market as Red Hat or Mandrake but has a devoted following; includes a large selection of applications; runs on Intel, PPC, Alpha, SPARC Itanium, mainframe, and other platforms (www.suse.com/index_us.html).

- **Red Hat Linux (8.0) —** is the most widely available and widely used Linux version by far, and many other distributions also use it as a base; its RPM software packaging solution solved a key difficulty in distribution and installation of Linux applications; lacks a disk partitioning utility; runs on Intel, Itanium, and Alpha platforms (www.redhat.com).
- **Debian GNU Linux (3.0) —** is a pure OSS version of Linux, using the Linux kernel with most of the basic OS tools from the GNU project; comes with more than 3,950 packages and precompiled software bundled up for easy installation; is known as one of the more advanced Linux distributions; maintained largely by volunteers; installation is difficult; runs on Intel, PPC, Alpha, SPARC, and other platforms (www.debian.org/index.en.html).
- **SCO Linux (4)** — derived from the Caldera Linux effort, SCO Linux is a product of the UnitedLinux effort, which includes Conectiva SA and Turbolinux Inc. and is led by SuSE Inc.; SCO Linux was released as a new product in November 2002 by the SCO Group; runs on Intel platforms (www.sco.com).
- **Linux Mandrake (9.0)** — was created in 1998 with the goal of making Linux easier to use for everyone; strong documentation is a special selling point, along with an included software selection utility and disk partition utility; Mandrake is based on Red Hat and comes with a large package selection, including StarOffice; runs on Intel platforms (www.mandrakelinux.com/en).
- **Turbolinux (8.0)** — a Linux version now specializing in Asian niche markets, with extra support for Asian languages; the company is currently experiencing some financial difficulties; runs on numerous platforms and environments (www.turbolinux.com).
- **Slackware (8.1) —** a highly advanced Linux operating system, designed with the twin goals of ease of use and stability as top priorities; initially runs right off the disk, without configuration utilities; runs on Intel, Alpha, and SPARC platforms (www.slackware.com).
- **LindowsOS (2.0)** — designed as “the Linux for Windows users,” it is an optimized Linux-based operating system that is designed to compete with MS Windows; originally designed to run Windows applications, it has now settled for locating open source Windows software equivalents; a “click and run” feature for easily adding new software is a key innovation; it costs more than other Linux distributions; runs on Intel platforms (www.lindows.com).

Total Cost of Ownership

Cost comparisons between different operating systems are often difficult, particularly so when the total cost of ownership (TCO) model is used. In real situations, the makeup of TCO is unique, depending upon an extremely wide variety of factors. Some of these factors include:

- Overall size of networks or systems in comparison
- Application mix
- Processor type, cost, and configuration
- Upgrade requirements
- Upgrade policy (and logic)
- Cost of administrators for the various environments (including local hiring costs and continuing training requirements)
- Number of systems per administrator and other tasks that an administrator can perform
- Efficiency of operation
- User training costs
- Software installation and configuration costs, including license counting and certification
- Excess costs added by license (Microsoft's new licensing scheme demands a license for every processor that *could run Windows* — which would also include most Linux units)
- Software maintenance cost
- Software upgrade cost
- General support costs

And then there are costs that may be saved or incurred by end users, whose professional time may or may not be spent handling system issues.

Several recent studies have fired off a new round of TCO comparisons between Microsoft and Linux and between Unix and Linux. A much-criticized December 2002 study by IDC, comparing Linux to Microsoft — and sponsored by Microsoft — gave Microsoft an advantage in TCO in three of four tested situations [3]. The problem with this study lay in the parameters provided by Microsoft. The major factor was in cost of labor, where Microsoft supplied the assumption that its trained professionals would be less expensive (due to the number available) and handle more machines. This naturally lowered Microsoft's costs. However, most studies in the actual workplace have demonstrated the opposite. First, Microsoft Certified Systems Administrators generally handle fewer boxes than their Linux counterparts. Second, Linux administrators can be significantly less expensive, particularly outside of the US, due to the widespread use of Linux and Unix within academic computing environments. Microsoft Certification also incurs high training costs, while Linux training tends to be less costly — neither was included in the study. Finally, the analysis was over a five-year term, while the update cycle for most corporations is three years — so the study missed the expensive sixth year upgrade "bomb."

A more realistic study was undertaken by Robert Frances Group in July 2002 using real-world conditions [14]. It compared Linux, Solaris, and Windows. The Solaris information was, perhaps, not entirely comparative since the processors involved were larger multiprocessor SPARC stations. However, between Microsoft and Windows, it tended to verify the consensus of the past several years — Windows is about twice as expensive as Linux in TCO. The results of this study are given in Table 4.

The extreme cost savings in moving from Unix to Linux, while heralded by the Linux community, are also problematic, due to basic differences in the equipment used and services being offered. An equally high (80%) TCO advantage for Linux over Unix was provided in a Red Hat IDC survey undertaken in late 2001 [6]. It may have been due to this study that Red Hat was more cautious about attacking IDC credibility for the Microsoft study.

A 2002 survey conducted by Cybersource compared Linux and Windows for TCO in a 250-workstation network [10]. It found that although salaries for staff running Linux are higher per annum ($376,000) compared to Microsoft ($345,000). The costs are more than offset by Microsoft licensing costs at $282,973.50 against Linux's sourcing costs, in this case, $879.95. The study assumed a three-year period. Linux also achieved savings through reduced downtime and improved performance. Results are given in Table 5.

Table 4 — Total Cost of Ownership for Linux, Solaris, and Windows [14]

Case	Year 1	Year 2	Year 3
Linux	$49,931	$62,203	$74,475
Solaris	$421,718	$491,619	$561,520
Windows	$91,724	$141,193	$190,662

Table 5 — Linux and Open Source Versus Microsoft TCO Comparison [10]

	Microsoft Solution TCO	Linux/Open Source TCO	Saving by Using Linux	Savings Percentage
Existing Hardware and Infrastructure Used	$733,973	$482,580	$251,393	34.26%
New Hardware and Infrastructure Purchased	$1,042,110	$790,717	$251,393	24.69%

TCO comparisons can, at best, only serve as guidelines for evaluation within a specific company and specific data processing situation. Although many such comparisons through the years have demonstrated lower costs for Linux, this should be taken as an opportunity to investigate rather than as a statement of fact or demonstrated cost benefit.

Case Studies

The following brief case studies illustrate how Linux and open source software have been successfully installed in a variety of situations, for benefits ranging from cost to suitability for special purpose applications.

Amazon.com, HP, and Linux

Source: Hewlett-Packard

Amazon.com implemented Linux across the enterprise in 2001. The primary motivations were cost savings and flexibility, although scalability and ease of use over comparable Unix solutions were also factors. The system was deployed by Hewlett-Packard, which needed to provide a smooth migration strategy to the Linux platform. Once deployed, the Linux system has proven flexible, stable, and economic.

In addition to other benefits, the Linux deployment resulted in a 20% reduction in technology and content costs in the first year.

Boeing, HP, and MSC.Software

Source: Hewlett-Packard

The R&D division at Boeing has replaced existing mainframe computing resources with Linux running on HP computer clusters. The primary motivation for the switch was a need to focus upon reducing costs. The division was using Cray supercomputers and IBM mainframes for complex computations to support aircraft design. It was thought possible to save costs by substituting Linux and Beowulf clusters. Beowulf (MSC.Software) is an approach to building a supercomputer as a cluster of off-the-shelf personal computers interconnected by a LAN.

The division bought a turnkey Beowulf cluster from MSC.Software, a company with which Boeing already had a relationship. MSC installed its own Linux version — MSC.Linux — which is optimized for compute-intensive applications. The initial cluster included 10 HP C36000 workstations running MSC.Linux. The entire system was deployed in two hours.

The Linux-based solution resulted in a 600% increase in performance and a cost reduction of 50%.

E*Trade, IBM, and Linux

Source: IBM

E*Trade integrates a diverse collection of investing, banking, lending, and financial planning services in a Web site presentation for customers. To achieve cost savings, the company transitioned to IBM xSeries servers running on Linux.

E*Trade can have 10,000 concurrent users from among its four million customer accounts, and its business depends upon providing easy access and rapid service. This requires a robust and scalable system capable of handling extreme peaks and valleys in customer interaction. The solution implemented included 90 xSeries 330 servers running Linux. Use of Linux has sharply reduced software licensing fees and service costs. An additional advantage is that the open architecture provides flexibility in adding new services and software.

NCSA, IBM, Linux Red Hat, and Turbolinux

Source: IBM

The National Computational Science Alliance, lead institution at the University of Illinois at Urbana-Champaign, provides a supercomputing environment for the science and engineering community, funded by the National Science Foundation. The institution provides its computing capacity through a Linux cluster solution.

NCSA worked with IBM to design the largest Linux cluster in the academic world. When completed, the cluster will be capable of a peak performance of two teraflops, tripling current supercomputer capability.

The configuration consists of one cluster with 512 IBM xSeries x300 thin servers, each with two Intel Pentium III processors running Red Hat Linux. A second cluster with 160 IBM IntelliStation 6894 Z Pro workstations, each with two Intel Itanium processors, is also being installed. Each cluster is interconnected with Myricom's Myrinet switch.

OMV, IBM, and Linux

OMV AG is one of Austria's largest industrial companies and one of Central and Eastern Europe's largest oil and gas groups. OMV is involved in exploration and production projects worldwide, operates an oil and gas supply system, and has a network of service stations across Central and Eastern Europe. The company had a SAP-based ERM solution that it needed to migrate to Linux for cost savings.

Five of OMV's SAP R/3 systems are now running either in whole or in part under Linux for IBM zSeries. All new SAP solutions — including mySAP CRM and SAP BW are being implemented on Linux for zSeries.

SAP solutions under Linux for zSeries have now been stable and successful since spring 2002.

Even the largest OMV SAP system, with about 2,500 users and a 350 GB database, uses applications servers powered by Linux.

Winnebago, IBM, and SuSE Linux

Source: IBM

Winnebago Industries needed to upgrade its e-mail system but was faced with an estimated $100,000 cost. It elected to replace the existing system with a Linux-based e-mail system on an IBM zSeries. Microsoft NT servers running Novell Netware and Microsoft Mail were replaced with a zSeries mainframe running SuSE Linux 7.0 and Bynari software. The Linux and Bynari solution solved e-mail, calendaring, and corruption problems, and it cost only one-third of other options. It has also permitted application consolidation on the zSeries.

Google and Red Hat Linux

Source: Red Hat

Google runs one of the most successful and high-profile search engines on the Web. At the heart of its operation is a suite of highly sophisticated searching technologies. The infrastructure used to run these services is one of the largest Linux clusters in the world, involving about 6,000 servers. The servers are unbranded Intel-based PCs running at 400-533 MHz.

From its prominent position as a search engine, Google traffic is extremely high. Hit rates reached about 14 million per day by mid-2000 and have continued to grow.

The Red Hat Linux solution was selected on the basis of cost. Given the 8,000 servers used, any other solution would have been far too expensive.

To provide optimal fault tolerance and server availability, the Linux cluster is built of subclusters, each comprising 200-300 servers. Data is replicated over several subclusters so the loss of machines does not diminish capacity. A consistent configuration makes it possible to quickly and easily reassign server roles.

Kenwood and Red Hat Linux

Source: Red Hat

Kenwood, a world leader in mobile, portable, and home audio electronics, was faced with the need to replace an aging McDonnell Douglas Information System (MDIS), which included a minicomputer with a Pick operating system deployed in the 1980s. The primary application required was an ERP system.

The solution selected was based on Red Hat Linux and jBASE software. Kenwood's Pick operating system-based source code was recompiled into native Linux executables. The resulting Linux-based solution solved the upgrade problem, as well as providing additional advantages, such as simplification of the desktop environment.

Following the success of its Red Hat Linux–based ERP system, Kenwood decided to replace its DOS-based POS system running its five Kenwood Factory Outlet stores. Previously running on a Windows NT or Windows 95 server in each store, the system has been redeployed on generic Pentium PCs running Red Hat Linux. The Linux solution permits file sharing via Samba and also acts as a firewall for the store's DSL Internet connection.

Roubaix General Hospital, Mandrake Linux

Source: MandrakeSoft

Roubaix General Hospital has 330 doctors and 2,700 agents; it provides 2,000 beds within the community of Lille, France. The original IT system was established in 1983 on a central IBM mainframe. Starting in 1996, this was migrated to an AIX Unix and Microsoft solution. The hospital recently developed a Linux extension for Web-based services that would coexist with the AIX/Microsoft environment.

To establish a Web presence, the hospital began an implementation under Windows, but found problems with speed in remote control applications and general reliability. A Linux solution was investigated and found to cost less than half of available alternatives. A Mandrake Linux solution was installed in five days with RAID. One server runs a firewall and DNS, a second provides an Apache Web server and Qmail e-mail router, and a third provides a proxy and internal DNS services.

The Linux service has been operating without failure for over a year. Benefits have included cost savings, plus the added bonus of reduced management requirements. Since AIX and Linux are similar, the two systems can be managed by a single team.

Conclusion

The general outlook for open source and Linux remains high, although growth has slightly tapered recently due to current economic conditions. Both open source and Linux have received significant endorsements across business and government users, as well as from major hardware and software vendors. Although presence in the enterprise is likely to remain small relative to Windows workstations, continued movement into the enterprise may reduce price expectations and provide some relief from the Microsoft monopoly.

As with many arguments that are phrased in black and white terms, open source versus conventional is not really an either/or situation. The fact that conventional software sales and distribution works is well demonstrated by the number of companies active in the area and the considerable profits that they have been able to achieve. It is also significant that the model for sales of both hardware and software has undergone considerable change over the years, with commoditization being a comparatively recent phenomenon. For many years, software has been sold on a yearly license/support arrangement. The PC brought about a commodity model, introducing a concept of “boxed software.” This market developed, but as these products became more useful within corporations, the original reasons for yearly licenses reemerged: namely, that software, particularly that used by novice users, requires support, customization, and bug fixes. Support requirements can actually exceed purchase costs.

Both open source and current commercial software models can coexist, and open source is likely to create pressure for lower costs, as well as continuing to develop an alternate pool for innovation.

One area that has shown considerable interest in open source is government. Those that have either adopted open source products or are experimenting with them include the governments of Mexico, Peru, South Korea, Thailand, Philippines, Australia, France, Germany, Taiwan, Pakistan, and China. Corporations are tentatively moving toward Linux implementation, particularly in the low-end server area — though this is certain to change as Linux scalability improves.

The US government has also shown an interest in open source, indirectly endorsing it in a number of contexts. For example, in 2000, the President's Information Technology Advisory Committee (PITAC) issued a report, "Developing Open Source Software to Advance High-End Computing," which issued recommendations favoring procurement of open source software. Though the PITAC recommendations were limited to research computing in a context, its arguments could be taken as generally favoring government use of open standards.

Although Linux is continuing to grow, there is some disagreement as to whether it is more of a challenge to existing Unix or to Windows. Morgan Stanley's August 2002 survey of 225 CIOs saw Linux continuing to move into IT budgets, with 29% of respondents saying they own Linux servers and 8% formally considering buying them [12]. Informally, 17% said they are considering Linux servers. For those who had recently purchased Linux servers, 31% were adding capacity, 31% were replacing Windows systems, and 24% were replacing Unix. The Butler Group predicted in October 2002 that Linux will continue to grow at the expense of commercial Unix, such as IBM's, HP's, and Sun's proprietary versions — with this trend being fed by increased certification of business applications on Linux. Most recently, the Linux growth spur has tapered a bit, although this may be a general result of recession. One thing certain is that it is now moving from the backwaters of IT technical centers into the mainstream.

An issue pushing some corporations to rethink policies is the fact that Microsoft is changing its licensing practices in ways that seem to result in increased costs to its customers. Some time ago, it changed licensing so that a single copy of Windows cannot be used for both home and office. Later, the company switched its large customers to a subscription-based approach called Licensing 6, which has had the overall effect of greatly increasing costs. In an ITIC/Sunbelt Software Microsoft Licensing survey, 80% of respondents had a negative view of the new licensing scheme, noting that costs for software assurance are now the highest in the industry [11]. Of those who had done a cost analysis, an overwhelming 90% said their costs would increase if they migrated to the 6.0 scheme. A full 38% of those surveyed said that they are actively seeking alternatives to Microsoft, although this is likely to be somewhat exaggerated.

Both OSS and Microsoft can summon up legions of followers. Accusations have passed both ways for years. Open software supporters argue that Microsoft is taking aim at them through a sophisticated disinformation program. Shortly after the publication of *The Cathedral and the Bazaar*, a number of internal Microsoft memos came to light, which were verified as to authenticity — but not as to importance — by Microsoft. These memos detailed a program of response whose ethics seemed murky, indeed, and reflected Microsoft's monopolistic position. These memos have since become known as the "Halloween Memos" for the period in which they were released. There are eight of them at the present time, and the collection appears to be growing [7].

Overall, the Halloween Memos demonstrate that Microsoft is beginning to take open source seriously. The rest of the information, although amusing to read, can be understood as any company's internal reaction to a perceived threat.

The open source movement is likely to continue into the enterprise, ushered in partly by Linux and partly by the fact that, after all, it is not so much different from the other ways that software has been developed and distributed in the relatively brief history of computing.

References

1. "2001 Linux Function Review." D.H. Brown Associates, September 2001.

2. Bailey, Michelle, Vernon Turner, Jean Bozman, and Janet Waxman. "Linux Servers: What's the Hype and What's the Reality?" IDC, March 2000.

3. Bozman, Jean, Al Gillen, Charles Kolodgy, Dan Kusnetzky, Randy Perry, and David Shiang. *Windows 2000 Versus Linux in Enterprise Computing: An Assessment of Business Value for Selected Workloads*. IDC White Paper (sponsored by Microsoft), December 2002.

4. *Free/Libre Open Source Software: Survey and Study*. Berlecon Research GmbH, July 2002 (www.infonomics.nl/FLOSS/report).

5. Gillen, Al, and Dan Kusnetzky. *Linux Overview: Understanding the Linux Market Model.* IDC, February 2000.

6. Gillen, Al, Dan Kusnetzky, and Scott McLarnon. *The Role of Linux in Reducing the Cost of Enterprise Computing.* IDC White Paper Sponsored by Red Hat, 2001 (www.redhat.com/whitepapers/services/tco.pdf).

7. *The Halloween Documents (Version 1.4).* Open Source Initiative, 2003 (http://opensource.org/halloween).

8. “IT Spending Survey United States.” Goldman Sachs, December 2002.

9. Kenwood, Carolyn A. *A Business Case Study of Open Source Software.* MITRE, July 2001 (www.mitre.org/support/papers/tech_papers_01/kenwood_software).

10. *Linux vs. Windows: Total Cost of Ownership Comparison*. Cybersource Pty. Ltd., 2002 (http://www.cyber.com.au/cyber/about/linux_vs_windows_tco_comparison.pdf).

11. “Microsoft Licensing 6.0 Survey.” ITIC/Sunbelt Software, March 2002.

12. Phillips, Charles, and Ryan Rathman. “Morgan Stanley CIO Survey Series, Release 3.6.” Morgan Stanley, August 2002.

13. Raymond, Eric S. *The Cathedral and the Bazaar.* O’Reilly & Associates, 2000.

14. *Total Cost of Ownership for Linux in the Enterprise.* Robert Frances Group, July 2002 (www-1.ibm.com/linux/RFG-LinuxTCO-vFINAL-Jul2002.pdf).

15. Weiss, George. “Study Finds Linux Meeting Increased Acceptance.” Datapro Research, February 1999.

Recommended Readings

Bezroukov, N. “Open Source Software Development as a Special Type of Academic Research.” *First Monday*, Vol. 4, No. 10, October 1999 (www.firstmonday.dk/issues/issue4_10/bezroukov/index.html).

Bezroukov, N. “A Second Look at the Cathedral and the Bazaar.” *First Monday*, Vol. 4, No. 12, December 1999 (www.firstmonday.dk/issues/issue4_12/bezroukov/index.html).

Free Software Foundation, GNU General Public License, v2, 1991 (www.gnu.org).

Gillen, Al, and Dan Kusnetzky. *Linux: A Journey into the Enterprise*. IDC White Paper Sponsored by Red Hat, 2001 (www.nebula.nl/downloads/Linux_in_the_Enterprise.pdf).

Linux in the Enterprise. Morse, 2002.

Prasad, C. Ganesh. *The Practical Manager’s Guide to Linux: Can You Profitably Use Linux in Your Organisation?* (www.osopinion.com/Opinions/GaneshCPrasad/GaneshCPrasad2.html).

Raymond, Eric S. “The Magic Cauldron.” 2000 (www.catb.org/~esr/writings/magic-cauldron).

Stevenson, Cooper. *Blueprint for Linux in the Enterprise*. Metasource Technologies, 17 June 2002 (www.metasource.us/linux.pdf).

About the Authors

Brian J. Dooley

Brian J. Dooley is an author, analyst, and journalist with more than 20 years' experience in analyzing and writing about trends in IT. He has written six books, numerous user manuals, hundreds of reports, and more than 2,000 magazine features. Mr. Dooley is the founder and past President of the New Zealand chapter of the Society for Technical Communication. He initiated and is on the board of the Graduate Certificate in Technical Communication program at Christchurch Institute of Technology, and he is on the editorial advisory board for Faulkner Technical Reports. Mr. Dooley currently resides in New Zealand, where he maintains a fully wired beach cottage and a Web site at http://bjdooley.com. He can be reached at bjd@bjdooley.com.

Marc R. Erickson

Now in his 30th year at IBM, Marc R. Erickson has been a key participant in several international innovations from IBM, including workflow technology service support center development, Lotus Notes standards, failure analysis and prediction technology, and establishment of IBM's embedded computing software technologies. Mr. Erickson is now on assignment as the communications manager for Eclipse, the open universal tools integration platform from the open source community founded in November 2001. Mr. Erickson can be reached at IBM Corporation, 601-109 Hutton Street, Raleigh, NC 27606, USA. E-mail: mre@us.ibm.com.

Gerald S. Greenberg

Gerald S. Greenberg is an original member of OSDL's board of directors. In his nearly 40 years in the computer industry, Mr. Greenberg has held senior marketing, product management, and engineering positions in Fujitsu's Unix and mainframe systems groups. He was a general manager and engineering manager at Data General and held management positions at Sperry Univac and the New York Stock Exchange. Most recently, he was Senior Vice President of Worldwide Marketing for Turbolinux. Mr. Greenberg has a bachelor's degree in mathematics from the City University of New York and a master's degree in computer and information sciences from the University of Pennsylvania. Mr. Greenberg can be reached at Open Source Development Lab, Inc., 15275 SW Koll Parkway, Suite H, Beaverton, OR 97006, USA. E-mail: jsg@osdl.org.

Luke Hohmann

Luke Hohmann is a management consultant committed to coaching his clients to greater levels of performance in the areas of product management, software development, and organizational effectiveness. Portions of his chapter in this report were based on material from his book *Beyond Software Architecture: Creating and Sustaining Winning Solutions* (Addison-Wesley, 2003). Mr. Hohmann can be reached at 599 Dawn Drive, Sunnyvale, CA 94087, USA. E-mail: luke@LukeHohmann.com.

Jason Matthews

Jason Matthews is a Senior Consultant with Cutter Consortium's Enterprise Architecture and Business-IT Strategies Practices. Mr. Matthews has more than 20 years of expertise in all aspects of business and technical management in IT companies. Previously, he was President/CEO of Genesis Development Corporation, a company he founded in 1987 to target the professional services sector of the then-nascent enterprise infrastructure software market. Mr. Matthews successfully led that business until the year 2000, when it was sold to IONA Technologies, PLC, a leading international software company. He then led the successful integration of the two companies, including meeting all financial requirements stemming from the merger, 17 months ahead of schedule. Mr. Matthews holds dual degrees in computer science and business administration from the University of Maryland. He is also the coauthor of the groundbreaking book *The Object Technology Revolution*, which correctly anticipated the rise of the Internet. He can be reached at jmatthews@cutter.com.

Michael Olson

Michael Olson, one of the original authors of Berkeley DB, is a technology industry veteran with more than 20 years' experience in engineering, marketing, sales, and business management. Mr. Olson was named president and CEO of Sleepycat in 2001 after serving as vice president of sales and marketing. Prior to Sleepycat, Mr. Olson served in technical and business management positions at database vendors Britton Lee, Illustra, and Informix. He holds bachelor's and master's degrees in computer science from the University of California at Berkeley. Mr. Olson can be reached at Sleepycat Software, 1509 McGee Avenue, Berkeley, CA 94703, USA. E-mail: mao@sleepycat.com.

Ganesh Prasad

Ganesh Prasad's background is in engineering, with postgraduate qualifications in computer science, management, and finance. Mr. Prasad has worked as a hands-on applications developer, analyst, and architect since 1987. His experience spans five generations of systems and platforms — mainframes with COBOL, proprietary minicomputers (VAX/VMS), Open Standard minicomputers (Unix with C), client-server systems (PowerBuilder on Windows), and Web technology. Having witnessed the complete demise of very popular proprietary systems (VAX, PowerBuilder), Mr. Prasad perceives a continuing long-term trend away from proprietary products and toward open standards and technologies. He follows his own advice to IT professionals to avoid proprietary, dead-end platforms such as Windows and .NET and to skill themselves in the technologies of the future — Linux and open source Java. Mr. Prasad lives in Sydney, Australia, and can be reached at E-mail: sashi@easy.com.au.

William A. Zucker

William A. Zucker is a Senior Consultant with Cutter Consortium's Risk Management Intelligence Network and Sourcing Practice and a frequent speaker at Cutter *Summits* and symposia. Mr. Zucker is a partner at Gadsby Hannah LLP, in Boston, Massachusetts, where he jointly heads the firm's litigation group and counsels clients on various business issues. His practice focuses on negotiation and litigation of business transactions, outsourcing and e-business, and technology/intellectual property. Mr. Zucker currently serves on the faculty of Norwich University, where he teaches the intellectual property aspects of computer security. He has authored or coauthored a number of publications, including: "The Legal Framework for Protecting Intellectual Property in the Field of Computing and Computer Software"; "IT Litigation Strategies"; "Outsourcing Do's and Don'ts"; "Negotiating the Outsourcing Relationship"; "Moving Beyond Traditional Dispute Resolution"; "Procurement Dialogues"; and "Don't Fence Me In: UCITA, A Wakeup Call for Software Users." Mr. Zucker is a member of the American Arbitration Association's National Technology Panel and the CPR Institute's working group on technology business alliances and conflict management. He is a graduate of Yale University and Harvard Law School. Mr. Zucker can be reached at Tel: +1 617 345 7016; E-mail: wzucker@cutter.com.

For More Information ...

Name	Title
Organization	Mail Stop
Address	E-Mail
City	State/Province
ZIP/Postal Code	Country
Telephone	Fax

☐ Please send me a complete listing of all current reports and directories for IT professionals.

Please send me more information on the following publications and services (check all that apply):

☐ ***Agile Project Management***	☐ ***Cutter IT Journal®***
☐ ***Business Intelligence***	☐ ***Enterprise Architecture***
☐ ***Business-IT Strategies***	☐ ***Risk Management Intelligence Network***
☐ ***Business Technology Trends and Impacts***	☐ ***Sourcing***
☐ ***Cutter Benchmark Review***	☐ ***Web Services Strategies®***

☐ Send me _____ additional copies of ***Open Source: Moving into the Enterprise*** for US $249 each (US $264 outside North America).

☐ Payment enclosed.

☐ Please charge my Visa, Mastercard, American Express, or Diners Club.
Charge will appear as Cutter Consortium.

Signature	
Card Number	Expiration Date

Mail to: Cutter Consortium
37 Broadway, Suite 1
Arlington, MA 02474-5552, USA

E-mail: sales@cutter.com

Call: +1 781 648 8700

Web site: www.cutter.com

Fax to: +1 781 648 1950

ISBN: 1-57484-182-3