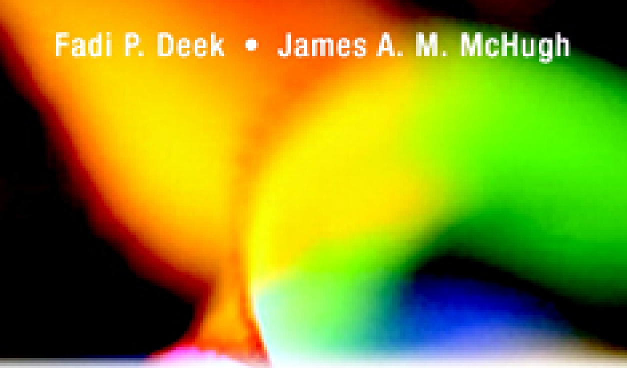
Fadi P. Deek • James A. M. McHugh



OPEN SOURCE

Technology and Policy

CAMBRIDGE www.cambridge.org/9780521881036

This page intentionally left blank

Open Source

From the Internet's infrastructure to operating systems like GNU/Linux, the open source movement comprises some of the greatest accomplishments in computing over the past quarter century. Its story embraces technological advances, unprecedented global collaboration, and remarkable tools for facilitating distributed development. The evolution of the Internet enabled an enormous expansion of open development, allowing developers to exchange information and ideas without regard to constraints of space, time, or national boundary. The movement has had widespread impact on education and government, as well as historic, cultural, and commercial repercussions. Part I discusses key open source applications, platforms, and technologies used in open development. Part II explores social issues ranging from demographics and psychology to legal and economic matters. Part III discusses the Free Software Foundation, open source in the public sector (government and education), and future prospects.

FADI P. DEEK received his Ph.D. in computer and information science from the New Jersey Institute of Technology (NJIT). He is Dean of the College of Science and Liberal Arts and Professor of Information Systems, Information Technology, and Mathematical Sciences at NJIT, where he began his academic career as a Teaching Assistant in 1985. He is also a member of the Graduate Faculty – Rutgers University Ph.D. Program in Management.

JAMES A. M. MCHUGH received his Ph.D. in applied mathematics from the Courant Institute of Mathematical Sciences, New York University. During the course of his career, he has been a Member of Technical Staff at Bell Telephone Laboratories (Wave Propagation Laboratory), Director of the Ph.D. program in computer science at NJIT, Acting Chair of the Computer and Information Science Department at NJIT, and Director of the Program in Information Technology. He is currently a tenured Full Professor in the Computer Science Department at NJIT.

Open Source

Technology and Policy

FADI P. DEEK
New Jersey Institute of Technology

JAMES A. M. McHUGH
New Jersey Institute of Technology

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, São Paulo

Cambridge University Press
The Edinburgh Building, Cambridge CB2 8RU, UK
Published in the United States of America by Cambridge University Press, New York

www.cambridge.org
Information on this title: www.cambridge.org/9780521881036

First published in print format 2007

ISBN-13 978-0-511-36412-9 eBook (Adobe Reader)
ISBN-10 0-511-36412-1 eBook (Adobe Reader)

ISBN-13 978-0-521-88103-6 hardback
ISBN-10 0-521-88103-X hardback

ISBN-13 978-0-521-70741-1 paperback
ISBN-10 0-521-70741-2 paperback

To my children,
Matthew, Andrew, and Rebecca

Fadi P. Deek

To my parents, Anne and Peter
To my family, Alice, Pete, and Jimmy
and to my sister, Anne Marie

James A. M. McHugh

Contents

Preface

The story of free and open software is a scientific adventure, packed with extraordinary, larger-than-life characters and epic achievements. From infrastructure for the Internet to operating systems like Linux, this movement involves some of the great accomplishments in computing over the past quarter century. The story encompasses technological advances, global software collaboration on an unprecedented scale, and remarkable software tools for facilitating distributed development. It involves innovative business models, voluntary and corporate participation, and intriguing legal questions. Its achievements have had widespread impact in education and government, as well as historic cultural and commercial consequences. Some of its attainments occurred before the Internet's rise, but it was the Internet's emergence that knitted together the scientific bards of the open source community. It let them exchange their innovations and interact almost without regard to constraints of space, time, or national boundary. Our story recounts the tales of major open community projects: Web browsers that fueled and popularized the Internet, the long dominant Apache Web server, the multifarious development of Unix, the near-mythical rise of Linux, desktop environments like GNOME, fundamental systems like those provided by the Free Software Foundation's GNU project, infrastructure like the X Window System, and more. We will encounter creative, driven scientists who are often bold, colorful entrepreneurs or eloquent scientific spokesmen. The story is not without its conflicts, both internal and external to the movement. Indeed the free software movement is perceived by some as a threat to the billions in revenue generated by proprietary firms and their products, or conversely as a development methodology that is limited in its ability to adequately identify consumer needs. Much of this tale is available on the Internet because of the way the community conducts its business, making it a uniquely accessible tale. As free and open software continues to increasingly permeate our private and professional lives, we believe this story will intrigue a wide audience of computer science students and practitioners, IT managers, policy-makers in government and education, and others who want to learn about the fabled, ongoing legacy of transparent software development.

Acknowledgments

Many people helped us during the process of writing and publishing this book. Although it is impossible to know all of them by name, we offer a word of appreciation and gratitude to all who have contributed to this project. In particular, we thank the anonymous reviewers who read the proposal for the text and carefully examined the manuscript during the earlier stages of the process. They provided excellent recommendations and offered superb suggestions for improving the accuracy and completeness of the presented material.

Heather Bergman, Computer Science Editor at Cambridge University Press, deserves enormous praise for her professionalism and competence. Heather responded promptly to our initial inquiry and provided excellent insight and guidance throughout the remaining stages. Her extraordinary efforts were instrumental in getting this book into the hands of its readers.

1
Introduction

The open source movement is a worldwide attempt to promote an open style of software development more aligned with the accepted intellectual style of science than the proprietary modes of invention that have been characteristic of modern business. The idea – or vision – is to keep the scientific advances created by software development openly available for everyone to understand and improve upon. Perhaps even more so than in the conventional scientific paradigm, the very process of creation in open source is highly transparent throughout. Its products and processes can be continuously, almost instantaneously scrutinized over the Internet, even retrospectively. Its peer review process is even more open than that of traditional science. But most of all: its discoveries are not kept secret and it lets anyone, anywhere, anytime free to build on its discoveries and creations.

Open source is transparent. The source code itself is viewable and available to study and comprehend. The code can be changed and then redistributed to share the changes and improvements. It can be executed for any purpose without discrimination. Its process of development is largely open, with the evolution of free and open systems typically preserved in repositories accessible via the Internet, including archives of debates on the design and implementation of the systems and the opinions of observers about proposed changes. Open source differs vastly from proprietary code where all these transparencies are generally lacking. Proprietary code is developed largely in private, albeit its requirements are developed with its prospective constituencies. Its source code is generally not disclosed and is typically distributed under the shield of binary executables. Its use is controlled by proprietary software licensing restrictions. The right to copy the program executables is restricted and the user is generally forbidden from attempting to modify and certainly from redistributing the code or possible improvements. In most respects, the two modalities of program development are polar opposites, though this is not to say there are not many areas where the commercial and open communities have cooperated.

Throughout this book, we will typically use the term *open source* in a generic sense, encompassing *free software* as referred to by the Free Software Foundation (FSF) and *open source software* as referred to by the Open Source Initiative (OSI) organization. The alternative composite terms *FLOSS* (for Free/Libre/Open Source Software) or *FOSS* are often used in a European context. The two organizations, the FSF and the OSI, represent the two streams of the free or open source movement. *Free software* is an intentionally evocative term, a rallying cry as it were, used by the FSF and intended to resonate with the values of freedom: user and developer freedom. The FSF's *General Public License* (GPL) is its gold standard for free licenses. It has the distinctive characteristic of preventing software licensed under it from being redistributed in a closed, proprietary distribution. Its motto might be considered as "share and share alike." However, the FSF *also* recognizes many other software licenses as *free* as long as they let the user run a program for any purpose, access its source code, modify the code if desired, and freely redistribute the modifications. The OSI on the other hand defines ten criteria for calling a license open source. Like the FSF's conditions for free software (though not the GPL), the OSI criteria do not require the software or modifications to be freely redistributed, allowing licenses that let changes be distributed in proprietary distributions. While the GPL is the free license preferred by the FSF, licenses like the (new) BSD or MIT license are more characteristic of the OSI approach, though the GPL is also an OSI-certified license. Much of the time we will not be concerned about the differences between the various kinds of free or open source licenses, though these differences can be very important and have major implications for users and developers (see such as Rosen, 2005). When necessary, we will make appropriate distinctions, typically referring to whether certain free software is GPL-licensed or is under a specific OSI-certified license. We will elaborate on software licenses in the chapter on legal issues. For convenience we will also refer at times to "open software" and "open development" in the same way.

We will begin our exploration by considering the rationale for open source, highlighting some of its putative or demonstrable characteristics, its advantages, and opportunities it provides. We will then overview what we will cover in the rest of the book.

1.1 Why Open Source

Before we embark on our detailed examination of open source, we will briefly explore some markers for comparing open and proprietary products. A proper comparison of their relative merits would be a massively complex, possibly infeasible undertaking. There are many perspectives that would have to be considered, as well as an immense range of products, operating in diverse settings, under different constraints, and with varied missions. Unequivocal data from unbiased sources would have to be obtained for an objective comparative evaluation, but this is hard to come by. Even for a single pair of open and proprietary products it is often difficult to come to clear conclusions about relative merits, except for the case of obviously dominant systems like Web servers (Apache). What this section modestly attempts is to set forth some of the parameters or metrics that can help structure a comparative analysis. The issues introduced here are elaborated on throughout the book.

Open source systems and applications often appear to offer significant benefits vis-à-vis proprietary systems. Consider some of the metrics they compete on. First of all, open source products are usually free of direct cost. They are often superior in terms of portability. You can modify the code because you can see it and it's allowed by the licensing requirements, though there are different licensing venues. The products may arguably be both more secure and more reliable than systems developed in a proprietary environment. Open products also often offer hardware advantages, with frequently leaner platform requirements. Newer versions can be updated to for free. The development process also exhibits potential macroeconomic advantages. These include the innately antimonopolistic character of open source development and its theoretically greater efficiency because of its arguable reduction of duplicated effort. The open source paradigm itself has obvious educational benefits for students because of the accessibility of open code and the development process' transparent exposure of high-quality software practice. The products and processes lend themselves in principle to internationalization and localization, though this is apparently not always well-achieved in practice. There are other metrics that can be considered as well, including issues of quality of vendor support, documentation, development efficiency, and so on. We will highlight some of these dimensions of comparison. A useful source of information on these issues is provided by the ongoing review at (Wheeler, 2005), a detailed discussion which, albeit avowedly sympathetic to the open source movement, makes an effort to be balanced in its analysis of the relative merits of open and proprietary software.

1.1.1 Usefulness, Cost, and Convenience

Does the open source model create useful software products in a timely fashion at a reasonable cost that are easy to learn to use? In terms of utility, consider that open source has been instrumental in transforming the use of computing in society. Most of the Internet's infrastructure and the vastly successful Linux operating system are products of open source style development. There are increasingly appealing open desktop environments like GNOME and KDE. Furthermore, many of these products like the early Web servers and browsers as well as Linux were developed quite rapidly and burst on the market. Firefox is a recent example. It is of course hard to beat the direct price of open source products since they are usually free. The zero purchase cost is especially attractive when the software product involved has already been commoditized. Commoditization occurs when one product is pretty much like another or at least good enough for the needs it serves. In such cases, it does not pay to pay more. An open source program like the Apache Web server does not even have to be best of breed to attract considerable market share; it just has to be cheap enough and good enough for the purpose it serves. Open source is also not only freely available but is free to update with new versions, which are typically available for free download on the same basis as the original. For most users, the license restrictions on open products are not a factor, though they may be relevant to software developers or major users who want to modify the products. Of course, to be useful, products have to be usable. Here the situation is evolving. Historically, many open source products have been in the category of Internet infrastructure tools or software used by system administrators. For such system applications, the canons of usability are less demanding because the users are software experts. For ordinary users, we observe that at least in the past interface, usability has not been recognized as a strong suit of open source. Open source advocate Eric Raymond observed that the design of desktops and applications is a problem of "ergonomic design and interface psychology, and hackers have historically been poor at it" (Raymond, 1999). Ease of installation is one aspect of open applications where usability is being addressed such as for the vendor-provided GNU/Linux distributions or, at a much simpler level, installers for software like the bundled AMP package (Apache, MySQL, Perl, PHP). (We use GNU/Linux here to refer to the combination of GNU utilities and the Linux kernel, though the briefer designation *Linux* is more common.) Another element in usability is user support. There is for-charge vendor-based support for many open source products just as is for proprietary products. Arguments have been made on both sides about which is better. Major proprietary software developers may have more financial resources to expend on "documentation, customer support and product training than do open source providers" (Hahn, 2002), but open source products by definition can have very wide networks of volunteer support. Furthermore, since the packages are not proprietary, the user is not locked-in to a particular vendor.

1.1.2 Performance Characteristics

Does open source provide products that are fast, secure, reliable, and portable? The overview in Wheeler (2005) modestly states that GNU/Linux is often either superior or at least competitive in performance with Windows on the same hardware environment. However, the same review emphasizes the sensitivity of performance to circumstances. Although proprietary developers benefit from financial resources that enable them to produce high quality software, the transparent character of open source is uniquely suitable to the requirements of security and reliability.

In terms of security, open source code is widely considered to be highly effective for mission-critical functions, precisely because its code can be publicly scrutinized for security defects. It allows users the opportunity to security-enhance their own systems, possibly with the help of an open source consultant, rather than being locked into a system purchased from a proprietary vendor (Cowan, 2003). In contrast, for example, Hoepman and Jacobs (2007) describe how the exposure of the code for a proprietary voting system revealed serious security flaws. Open accessibility is also necessary for government security agencies that have to audit software before using it to ensure its operation is transparent (Stoltz, 1999). Though security agencies can make special arrangements with proprietary distributors to gain access to proprietary code, this access is automatically available for open source. Open source products also have a uniquely broad peer review process that lends itself to detection of defects during development, increasing reliability. Not only are the changes to software proposed by developers scrutinized by project maintainers, but also any bystander observing the development can comment on defects, propose implementation suggestions, and critique the work of contributors. One of the most well-known aphorisms of the open source movement "Given enough eyeballs, all bugs are shallow" (Raymond, 1998) identifies an advantage that may translate into more reliable software. In open source "All the world's a stage" with open source developers very public actors on that stage. The internal exposure and review of open source occurs not just when an application is being developed and improvements are reviewed by project developers and maintainers, but for the entire life cycle of the product because its code is always open. These theoretical benefits of open source appear to be verified by data. For example, a significant empirical study described in Reasoning Inc. (2003) indicates that free MySQL had six times fewer defects than comparable proprietary databases (Tong, 2004). A legendary acknowledgment of Linux reliability was presented in the famous Microsoft Halloween documents (Valloppillil, 1998) which described Linux as having a failure rate two to five times lower than commercial Unix systems.

The open source Linux platform is the most widely ported operating system. It is dominant on servers, workstations, and supercomputers and is widely used in embedded systems like digital appliances. In fact, its portability is directly related to the design decisions that enabled the distributed open style of development under which Linux was built in the first place. Its software organization allowed architect Linus Torvalds to manage core kernel development while other distributed programmers could work independently on so-called kernel modules (Torvalds, 1999). This structure helped keep hardware-specific code like device drivers out of the core kernel, keeping the core highly portable (Torvalds, 1999). Another key reason why Linux is portable is because the GNU GCC compiler itself is ported to most "major chip architectures" (Torvalds, 1999, p. 107). Ironically, it is the open source Wine software that lets proprietary Windows applications portably run on Linux. Of course, there are open source clones of Windows products like MS Office that work on Windows platforms. A secondary consideration related to portability is software localization and the related notion of internationalization. Localization refers to the ability to represent a system using a native language. This can involve the language a system interface is expressed in, character-sets or even syntactical effects like tokenization (since different human languages are broken up differently, which can impact the identification of search tokens). It may be nontrivial for a proprietary package that is likely to have been developed by a foreign corporation to be localized, since the corporate developer may only be interested in major language groupings. It is at least more natural for open software to be localized because the source code is exposed and there may be local open developers interested in the adaptation. Internationalization is a different concept where products are designed in the first place so that they can be readily adapted, making subsequent localization easier. Internationalization should be more likely to be on the radar screen in an open source framework because the development model itself is international and predisposed to be alert to such concerns. However, Feller and Fitzgerald (2002) who are sympathetic to free software critique it with respect to internationalization and localization, contrasting what appears to be, for example, the superior acceptability of the Microsoft IIS server versus Apache on these metrics. They suggest the root of the problem is that these characteristics are harder to "achieve if they are not factored into the original design" (p. 113). Generally, open source seems to have an advantage in supporting the customization of applications over proprietary code, because its code is accessible and modification of the code is allowed by the software license.

1.1.3 Forward-looking Effects

Is open source innovative or imitative? The answer is a little of both. On the one hand, open source products are often developed by imitating the functionality of existing proprietary products, "following the taillights" as the saying goes. This is what the GNOME project does for desktop environments, just like Apple and Microsoft took off on the graphical environments developed at Xerox PARC in the early 1980s. However, open development has also been incredibly innovative in developing products for the Internet environment, from infrastructure software like code implementing the TCP/IP protocols, the Apache Web server, the early browsers at CERN and NCSA that led to the explosion of commercial interest in the Internet to hugely successful peer-to-peer file distribution software like BitTorrent. Much of the innovation in computing has traditionally emerged from academic and governmental research organizations. The open source model provides a singularly appropriate outlet for deploying these innovations: in a certain sense it keeps these works public.

In contrast, Microsoft, the preeminent proprietary developer, is claimed by many in the open community to have a limited record of innovation. A typical contention is illustrated in the claim by the FSF's Moglen that "Microsoft's strategy as a business was to find innovative ideas elsewhere in the software marketplace, buy them up and either suppress them or incorporate them in its proprietary product" (Moglen, 1999). Certainly a number of Microsoft's signature products have been reimplementations of existing software (Wheeler, 2006) or acquisitions which were possibly subsequently improved on. These include QDOS (later MS-DOS) from Seattle Computer in 1980 (Conner, 1998), FrontPage from Vermeer in 1996 (Microsoft Press Release, 1996), PowerPoint from Forethought in 1987 (Parker, 2001), and Cooper's Tripod subsequently developed at Microsoft into Visual Basic in 1988 (Cooper, 1996). In a sense, these small independent companies recognized opportunities that Microsoft subsequently appropriated. For other examples, see McMillan (2006). On the other hand, other analysts counter that a scenario where free software dominated development could seriously undermine innovation. Thus Zittrain (2004) critically observes that "no one can readily monopolize derivatives to popular free software," which is a precondition to recouping the investments needed to improve the original works; see also Carroll (2004).

Comparisons with proprietary accomplishments aside, the track record on balance suggests that the open source paradigm encourages invention. The availability of source code lets capable users play with the code, which is a return to a venerable practice in the history of invention: tinkering (Wheeler, 2005).

The public nature of Internet-based open development provides computer science students everywhere with an ever-available set of world-class examples of software practice. The communities around open source projects offer unique environments for learning. Indeed, the opportunity to learn is one of the most frequently cited motivations for participating in such development. The model demonstrably embodies a participatory worldwide engine of invention.

1.1.4 Economic Impact

Free and open software is an important and established feature of the commercial development landscape. Granted, no open source company has evolved to anything like the economic status of proprietary powerhouses like Microsoft; nonetheless, the use of open source, especially as supporting infrastructure for proprietary products, is a widely used and essential element of the business strategies of major companies from IBM to Apple and Oracle. Software companies traditionally rely at least partly on closed, proprietary code to maintain their market dominance. Open source, on the other hand, tends to undermine monopoly, the likelihood of monopolistic dominance being reduced to the extent that major software infrastructure systems and applications are open. The largest proprietary software distributors are U.S. corporations – a factor that is increasingly encouraging counterbalancing nationalistic responses abroad. For example, foreign governments are more than ever disposed to encourage a policy preference for open source platforms like Linux. The platforms' openness reduces their dependency on proprietary, foreign-produced code, helps nurture the local pool of software expertise, and prevents lock-in to proprietary distributors and a largely English-only mode where local languages may not even be supported. Software is a core component of governmental operation and infrastructure, so dependency on extranational entities is perceived as a security risk and a cession of control to foreign agency.

At the macroeconomic level, open source development arguably reduces duplication of effort. Open code is available to all and acts as a public repository of software solutions to a broad range of problems, as well as best practices in programming. It has been estimated that 75% of code is written for specific organizational tasks and not shared or publicly distributed for reuse (Stoltz, 1999). The open availability of such source code throughout the economy would reduce the need to develop applications from scratch. Just as software libraries and objects are software engineering paradigms for facilitating software reuse, at a much grander scale the open source movement proposes to preserve entire ecosystems of software, open for reuse, extension, and modification. It has traditionally been perceived that "open source software is often geared toward information technology specialists, to whom the availability of source code can be a real asset, (while) proprietary software is often aimed at less sophisticated users" (Hahn, 2002). Although this observation could be refined, generally a major appeal of open source has been that its code availability makes it easier for firms to customize the software for internal applications. Such in-house customization is completely compatible with all open source licenses and is extremely significant since most software is actually developed or custom-designed rather than packaged (Beesen, 2002). As a process, open source can also reduce the development and/or maintenance risks associated with software development even when done by private, for-profit companies. For example, consider code that has been developed internally for a company. It may often have little or no external sales value to the organization, even though it provides a useful internal service. Stallman (1999) recounts the example of a distributed print-spooler written for an in-house corporate network. There was a good chance the life cycle of the code would be longer than the longevity of its original programmers. In this case, distributing the code as open source created the possibility of establishing an open community of interest in the software. This is useful to the company that owns the code since it reduces the risk of maintenance complications when the original developers depart. With any luck, it may connect the software to a persistent pool of experts who become familiar with the software and who can keep it up to date for their own purposes. More generally, open development can utilize developers from multiple organizations in order to spread out development risks and costs, splitting the cost among the participants. In fact, while much open source code has traditionally been developed with a strong volunteer pool, there has also been extensive industrial support for open development. Linux development is a prime example. Developed initially under the leadership of Linus Torvalds using a purely volunteer model, most current Linux code contributions are done by professional developers who are employees of for-profit corporations.

References

Beesen, J. (2002). What Good is Free Software? In: Government Policy toward Open Source Software, R.W. Hahn (editor). Brookings Institution Press, Washington, DC.

Carroll, J. (2004). Open Source vs. Proprietary: Both Have Advantages. ZDNet Australia. http://opinion.zdnet.co.uk/comment/0,1000002138,39155570,00.htm. Accessed June 17, 2007.

Conner, D. (1998). Father of DOS Still Having Fun at Microsoft, Microsoft MicroNews, April 10. http://www.patersontech.com/Dos/Micronews/paterson04_10_98.htm. Accessed December 20, 2006.

Cooper, A. (1996). Why I Am Called "the Father of Visual Basic," Cooper Interaction design. http://www.cooper.com/alan/father_of_vb.html. Accessed December 20, 2006.

Cowan, C. (2003). Software security for open-source systems. *IEEE Security and Privacy*, 1, 38–45.

Feller, J. and Fitzgerald, B. (2002). Understanding Open Source Software Development. Addison-Wesley, Pearson Education Ltd., London.

Hahn, R. (2002). Government Policy toward Open Source Software: An Overview. In: Government Policy toward Open Source Software, R.W. Hahn (editor). Brookings Institution Press, Washington, DC.

Hoepman J.H. and Jacobs, B. (2007). Increased Security through Open Source, *Communications of the ACM*, 50(1), 79–83.

McMillan, A. (2006). Microsoft "Innovation." http://www.mcmillan.cx/innovation.html. Accessed December 20, 2006.

Microsoft Press Release. (1996). Microsoft Acquires Vermeer Technologies Inc., January 16th. http://www.microsoft.com/presspass/press/1996/jan96/vrmeerpr.mspx. Accessed December 20, 2006.

Moglen, E. (1999). Anarchism Triumphant: Free Software and the Death of Copyright. First Monday, 4(8). http://www.firstmonday.org/issues/issue4_8/moglen/index.html. Accessed January 5, 2007.

Parker, I. (2001). Absolute Powerpoint – Can a Software Package Edit Our Thoughts. New Yorker, May 28. http://www.physics.ohio-state.edu/~wilkins/group/powerpt.html. Accessed December 20, 2006.

Raymond, E. (1999). The Revenge of the Hackers. In: Open Sources: Voices from the Open Source Revolution, M. Stone, S. Ockman, and C. DiBona (editors). O'Reilly Media, Sebastopol, CA, 207–219.

Raymond, E.S. (1998). The Cathedral and the Bazaar. *First Monday*, 3(3). http://www.firstmonday.dk/issues/issue3_3/raymond/index.html. Accessed December 3, 2006.

Reasoning Inc. (2003). How Open Source and Commercial Software Compare: MySQL white paper MySQL 4.0.16. http://www.reasoning.com/downloads.html. Accessed November 29, 2006.

Rosen, L. (2005). Open Source Licensing: Software Freedom and Intellectual Property Law, Prentice Hall, Upper Saddle River, NJ.

Stallman, R. (1999). The Magic Cauldron. http://www.catb.org/esr/writings/magic-cauldron/. Accessed November 29, 2006.

Stoltz, M. (1999). The Case for Government Promotion of Open Source Software. NetAction White Paper. http://www.netaction.org/opensrc/oss-report.html. Accessed November 29, 2006.

Tong, T. (2004). Free/Open Source Software in Education. United Nations Development Programme's Asia-Pacific Information Programme, Malaysia.

Torvalds, L. (1999). The Linux Edge. In: Open Sources: Voices from the Open Source Revolution, M. Stone, S. Ockman, and C. DiBona (editors). O'Reilly Media, Sebastopol, CA, 101–112.

Valloppillil, V. (1998). Open Source Software: A (New?) Development Methodology. http://www.opensource.org/halloween/. The Halloween Documents. Accessed November 29, 2006.

Wheeler, D. (2005). Microsoft the Innovator? http://www.dwheeler.com/innovation/microsoft.html. Accessed November 29, 2006.

Wheeler, D. (2006). Why Open Source Software/Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers! http://www.dwheeler.com/oss_fs_why.html. Accessed November 29, 2006.

Zittrain, J. (2004). Normative Principles for Evaluating Free and Proprietary Software. University of Chicago Law Review, 71(1), 265–287.

1.2 Preview

We will view the panorama of open source development through a number of different lenses: brief descriptive studies of prominent projects, the enabling technologies of the process, its social characteristics, legal issues, its status as a movement, business venues, and its public and educational roles. These perspectives are interconnected. For example, technological issues affect how the development process works. In fact, the technological tools developed by open source projects have at the same time enabled its growth. The paradigm has been self-hosting and self-expanding, with open systems like Concurrent Versions System (CVS) and the Internet vastly extending the scale on which open development takes place. Our case studies of open projects will reveal its various social, economic, legal, and technical dimensions. We shall see how its legal matrix affects its business models, while social and psychological issues are in turn affected by the technological medium. Though we will separate out these various factors, the following chapters will also continually merge these influences. The software projects we consider are intended to familiarize the reader with the people, processes, and accomplishments of free and open development, focusing on Internet applications and free software platforms. The enabling technologies of open development include the fascinating versioning systems both centralized and distributed that make enormous open projects feasible. Such novel modes of collaboration invariably pose new questions about the social structures involved and their affect on how people interact, as well as the psychological and cognitive phenomena that arise in the new medium/modality. Open development is significantly dependent on a legal infrastructure as well as on a technological one, so we will examine basic legal concepts including issues like licensing arrangements and the challenge of software patents. Social phenomena like open development do not just happen; they depend on effective leadership to articulate and advance the movement. In the case of free and open software, we shall see how the FSF and the complementary OSI have played that role. The long-term success of a software paradigm requires that it be economically viable. This has been accomplished in free software in different ways, from businesses based purely on open source to hybrid arrangements more closely aligned with proprietary strategies. Beyond the private sector, we consider the public sector of education and government and how they capitalize on open source or affect its social role. We will close our treatment by briefly considering likely future developments, in a world where information technology has become one of the central engines of commerce and culture.

Section One of the book covers key open source Internet applications and platforms, and surveys technologies used in distributed collaborative open development. *Section Two* addresses social issues ranging from the demographics of participants to legal issues and business/economic models. *Section Three* highlights the role of the Free Software Foundation in the movement, the relation of open source to the public sector in government and education, and future prospects. A glimpse of the topics covered by the remaining chapters follows.

Chapter 2 recounts some classic stories of open development related to the Internet, like Berners-Lee's groundbreaking work on the Web at CERN, the development of the NCSA HTTP Web server and Mosaic browser, the Apache project, and more. These case studies represent remarkable achievements in the history of business and technology. They serve to introduce the reader unfamiliar with the world of open source to some of its signature projects, ideas, processes, and people. The projects we describe have brought about a social and communications revolution that has transformed society. The story of these achievements is instructive in many ways: for learning how the open source process works, what some of its major attainments have been, who some of the pioneering figures in the field are, how projects have been managed, how people have approached development in this context, what motivations have led people to initiate and participate in such projects, and some of the models for commercialization. We consider the servers and browsers that fueled the Internet's expansion, programming languages like Perl and PHP and the MySQL database so prominent in Internet applications, newer systems like BitTorrent, Firefox, and others. We also review the Fetchmail project that became famous as an exemplar of Internet-based, collaborative, bazaar-style development because of a widely influential essay.

Chapter 3 explores the open source platform by which we mean the open operating systems and desktops that provide the infrastructure for user interaction with a computer system. The root operating system model for open source was Unix. Legal and proprietary issues associated with Unix led to the development of the fundamentally important free software GNU project, the aim of which was to create a complete and self-contained free platform that would allow anyone to do all their software development in a free software environment. The flagship Linux operating system evolved out of a port of a Unix variant to a personal computer environment and then burgeoned into the centerpiece project of the open software movement. The Linux and free Unix-like platforms in turn needed a high-quality desktop style interface and it was out of this imperative that the two major open desktops GNOME and KDE emerged, which in turn depended on the fundamental functionality provided by the X Window System. This chapter recounts these epic developments in the history of computing, describing the people, projects, and associated technical and legal issues.

Chapter 4 overviews the key technologies used to manage open source projects, with a special emphasis on CVS. The free software movement emerged in the early 1980s, at a time when the ARPANET network with its several hundred hosts was well-established and moving toward becoming the Internet. The ARPANET allowed exchanges like e-mail and FTP, technologies that significantly facilitated distributed collaboration, though the Internet was to greatly amplify this. The TCP/IP protocols that enabled the Internet became the ARPANET standard on January 1, 1983, about the same time the flagship open source GNU project was announced by free software leader and advocate Richard Stallman. By the late 1980s the NSFNet backbone network merged with the ARPANET to form the emerging worldwide Internet. The exponential spread of the Internet catalyzed further proliferation of open development. The specific communications technologies used in open source projects have historically tended to be relatively lean: e-mail, mailing lists, newsgroups, and later on Web sites, Internet Relay Chat, and forums. Major open source projects like Linux in the early 1990s still began operation with e-mail, newsgroups, and FTP downloads to communicate. Newsgroups provided a means to broadcast ideas to targeted interest groups whose members might like to participate in a development project. Usenet categories acted like electronic bulletin boards which allowed newsgroup participants to post e-mail-like messages, like the famous comp.os.minix newsgroup on Usenet used by Linus Torvalds to initiate the development of Linux. A powerful collaborative development tool was developed during the late 1980s and early 1990s that greatly facilitated managing distributed software development: the versioning system. Versioning systems are software tools that allow multiple developers to work on projects concurrently and keep track of changes made to the code. This chapter describes in some detail how CVS works. To appreciate what it does it is necessary to have a sense of its commands, their syntax, and outputs or effects and so we examine these closely. We also consider newer versioning tools like the decentralized system *BitKeeper* that played a significant role in the Linux project for a period of time, its free successor *Git*, and the *Subversion* system. Other means that have facilitated open source development have been the software hosting facilities that help distributed collaborators manage their open source projects and provide source code repositories for projects. We describe some of the services they provide and the major Web sites.

There are many demographic, social, psychological, cognitive, process, and media characteristics that affect open source development. Chapter 5 overviews some of these. It also introduces a variety of concepts from the social sciences that can be brought to bear on the open source phenomenon to help provide a framework for understanding this new style of human, scientific, and commercial interaction. We first of all consider the basic demographics of the phenomenon, such as the number and scale of projects under development, the kinds of software that tend to be addressed, population characteristics and motivations for developers and community participants, how participants interact. We survey relevant concepts from social psychology, including the notions of norms and roles, the factors that affect group interactions like compliance, internalization, and identification, normative influences, the impact of power relationships, and group cohesion. Ideas like these from the field of social psychology help provide conceptual tools for understanding open development. Other useful abstractions come from cognitive psychology, like the well-recognized cognitive biases that affect group interactions and problem solving. Social psychology also provides models for understanding the productivity of collaborative groups in terms of what are called process losses and gains, as well as organizational effects that affect productivity. The impact of the collaborative medium on group interactions is worth understanding, so we briefly describe some of the classic research on the effect of the communications medium on interaction. Like the field of social psychology, media research offers a rich array of concepts and a point of departure for understanding and analyzing distributed collaboration. Potentially useful concepts range from the effect of so-called common ground, coupling, and incentive structures, to the use of social cues in communication, the richness of informational exchanges, and temporal effects in collaboration. We introduce the basic concepts and illustrate their relevance to open collaboration.

The open source movement is critically affected by legal issues related to intellectual property. Intellectual property includes creations like copyrighted works, patented inventions, and proprietary software. The objective of Chapter 6 is to survey the related legal issues in a way that is informative for understanding their impact on free and open development. In addition to copyright and patent, we will touch on topics like software patents, licenses and contracts, trademarks, reverse engineering, the notion of reciprocity in licensing, and derivative works in software. The legal and business mechanisms to protect intellectual property are intended to address what is usually considered to be its core problem: how to protect creations in order to provide incentives for innovators. Traditionally such protection has been accomplished through exclusion. For example, you cannot distribute a copyrighted work for your own profit without the authorization of the copyright owner. The FSF's GPL that lies at the heart of the free software movement takes a very different attitude to copyright, focusing not on how to invoke copyright to exclude others from using your work, but on how to apply it to preserve the free and open distribution of your work, particularly when modified. We describe the GPL and the rationales for its conditions. We also consider the OSI and the motivations for its licensing criteria. The OSI, cofounded by Eric Raymond and Bruce Perens in 1998, was established to represent what was believed to be a more pragmatic approach to open development than that championed by the FSF. The OSI reflected the experience of the stream of the free software movement that preferred licenses like the BSD and MIT licenses which appeared more attractive for commercial applications. It reflected the attitude of developers like McKusick of the BSD project and Gettys of the X Window System. We describe some of the OSI-certified software licenses including the increasingly important Mozilla Public License. We also briefly address license enforcement and international issues, and the status and conditions of the next version of the GPL: GPLv3.

Chapter 7 examines economic concepts relevant to open source development, the basic business models for open products, the impact of software commoditization, and economic models for why individuals participate in open development. Some of the relevant economic concepts include vendor lock-in, network effects (or externalities), the total cost of use of software, the impact of licensing on business models, complementary products, and the potential for customizability of open versus proprietary products. The basic open business models we describe include dual licensing, consultation on open source products, provision of open source software distributions and related services, and the important hybrid models like the use of open source for in-house development or horizontally in synergistic combination with proprietary products, such as in IBM's involvement with Apache and Linux. We also examine software commoditization, a key economic phenomenon that concerns the extent to which a product's function has become commoditized (routine or standard) over time. Commoditization deeply affects the competitive landscape for proprietary products. We will present some of the explanations that have been put forth to understand the role of this factor in open development and its implications for the future. Finally, observers of the open source scene have long been intrigued by whether developers participate for psychological, social, or other reasons. We will consider some of the economic models that have been offered to explain why developers are motivated to work on these projects. One model, based on empirical data from the Apache project, uses an effect called signaling to explain why individuals find it economically useful to volunteer for open source projects. Another model proposes that international differences in economic conditions alter the opportunity cost of developer participation, which in turn explains the relative participation rates for different geographic regions.

The chapter on legal issues recounted the establishment and motivation for the OSI in 1998 and Chris Peterson's coinage of the *open source* designation as an alternative to what was thought to be the more ideologically weighted phrase *free software*. The OSI represents one main stream of the open software movement. Of course, the stream of the movement represented by the FSF and the GNU project had already been formally active since the mid-1980s. The FSF and its principals, particularly Richard Stallman, initiated the free software concept, defined its terms, vigorously and boldly publicized its motivations and objectives, established the core GNU project, and led advocacy for the free software movement. They have been instrumental in its burgeoning success. Chapter 8 goes into some detail to describe the origin and technical objectives of the GNU project, which represents one of the major technical triumphs of the free software movement. It also elaborates on the philosophical principles espoused by the FSF, as well as some of the roles and services the FSF provides.

Chapter 9 considers the role of open source in the public sector which, in the form of government and education, has been critical to the creation, development, funding, deployment, and promotion/advocacy of open software. The public sector continues to offer well-suited opportunities for using and encouraging open source, in domains ranging from technological infrastructure to national security, educational use, administrative systems, and so on, both domestically and internationally. Open source has characteristics that naturally suit many of these areas. Consider merely the role of the public sector in supporting the maintenance and evolution of technological infrastructure for society, an area in which open software has proven extremely successful. The government has also historically played an extensive role in promoting innovation in science and technology. For example, the federal government was the leader in funding the development of the Internet with its myriad of underlying open software components. Thus public investment in open development has paid off dramatically in the past and can be expected to continue to do so in the future. The transparency of open source makes it especially interesting in national security applications. Indeed, this is an increasingly recognized asset in international use where proprietary software may be considered, legitimately or not, as suspect. Not only do governmental agencies benefit as users of open source, government and educational institutions also play a role in promoting its expanded use. Governmental policy decisions, whether of a legislative or policy-driven character, can significantly affect the expansion of open software use in the government and by the public. For example, nationalistic concerns about the economic autonomy of local software industries or about national security have made open source increasingly attractive in the international arena. Lastly, we will address at some length the uses and advantages of open source in education, including its unique role in computer science education.

We conclude our book in Chapter 10 with what, we believe, are the likely scenarios for the prospective roles of open and proprietary software. Our interpretation is a balanced one. On the one hand, the open source paradigm seems likely to continue its advance toward worldwide preeminence in computer software infrastructure, not only in the network and its associated utilities, but also in operating systems, desktop environments, and standard office utilities. Significantly, the most familiar and routine applications seem likely to become commoditized and open source, resulting in pervasive public recognition of the movement. The software products whose current dominance seems likely to decline because of this transformation include significant parts of the current Microsoft environment from operating systems to office software. However, despite a dramatic expansion in the recognition and use of open source, this in no ways means that open software will be dominant in software applications. To the contrary, the various dual modalities that have already evolved are likely to persist, with robust open and proprietary sectors each growing and prevailing in different market domains. While on the one hand, some existing proprietary systems may see portions of their markets overtaken by open source replacements, on the other hand proprietary applications and hybrid modes of commercial development should continue to strengthen. Specialized proprietary killer-apps serving mega-industries are likely to continue to dominate their markets, as will distributed network services built on open infrastructures that have been vertically enhanced with proprietary functionalities. Mixed application modes like those reflected in the WAMP stack (with Windows used in place of Linux in the LAMP stack) and the strategically significant Wine project that allows Windows applications to run on Linux environments will also be important. The nondistributed, in-house commercial development that has historically represented the preponderance of software development seems likely to remain undisclosed either for competitive advantage or by default, but this software is being increasingly built using open source components – a trend that is already well-established. The hybrid models that have emerged as reflected in various industrial/community cooperative arrangements, like those involving the Apache Foundation, the X Window System, and Linux, and based on industrial support for open projects under various licensing arrangements, seem certain to strengthen even further. They represent an essential strategy for spreading the risks and costs of software development and providing an effective complementary set of platforms and utilities for proprietary products.

SECTION ONE

Open Source – Internet Applications, Platforms, and Technologies

2

Open Source Internet Application Projects

This chapter describes a number of open source applications related to the Internet that are intended to introduce the reader unfamiliar with the world of open development to some of its signature projects, ideas, processes, and people. These projects represent remarkable achievements in the history of technology and business. They brought about a social and communications revolution that transformed society, culture, commerce, technology, and even science. The story of these classic developments as well as those in the next chapter is instructive in many ways: for learning how the open source process works, what some of its major accomplishments have been, who some of the pioneering figures in the field are, how projects have been managed, how people have approached development in this context, what motivations have led people to initiate and participate in such projects, and what some of the business models are that have been used for commercializing associated products.

Web servers and Web browsers are at the heart of the Internet and free software has been prominent on both the server and browser ends. Thus the first open source project we will investigate is a server, the so-called National Center for Supercomputing Applications (NCSA) Web server developed by Rob McCool in the mid-1990s. His work had in turn been motivated by the then recent creation by Tim Berners-Lee of the basic tools and concepts for a World Wide Web (WWW), including the invention of the first Web server and browser, HTML (the Hypertext Markup Language), and the HTTP (Hypertext Transfer Protocol). For various reasons, McCool's server project subsequently forked, leading to the development of the Apache Web server. It is instructive and exciting to understand the dynamics of such projects, the contexts in which they arise, and the motivations of their developers. In particular, we will examine in some detail how the Apache project emerged, its organizational processes, and what its development was like. Complementary to Web servers, the introduction of easily used Web browsers had an extraordinary impact on Web use, and thereby a revolutionary effect on business, technology, and society at large. The Mosaic, Netscape, and more recently the Firefox browser projects that we will discuss even shared some of the same development context. The success of the Mosaic browser project was especially spectacular. In fact it was instrumental in catalyzing the historic Internet commercial revolution. Mosaic's developer Marc Andreessen later moved on to Netscape, where he created, along with a powerhouse team of developers, the Netscape browser that trumped all competition in the browser field for several years. But Netscape's stunning success proved to be temporary. After its initial triumph, a combination of Microsoft's bundling strategies for Internet Explorer (IE) and the latter's slow but steady improvement eventually won the day over Netscape. Things lay dormant in the browser area for a while until Firefox, a descendant of the Netscape Mozilla browser, came back to challenge IE, as we shall describe.

The process of computer-supported, distributed collaborative software development is relatively new. Although elements of it have been around for decades, the kind of development seen in Linux was novel. Eric Raymond wrote a famous essay on Linux-like development in which he recounted the story of his own Fetchmail project, an e-mail utility. Although Fetchmail is far less significant as an open source product than other projects that we review, it has come to have a mythical *pedagogical* status in the field because Raymond used its development – which he intentionally modeled on that of Linux – as an exemplar of how distributed open development works and why people develop software this way. Raymond's viewpoints were published in his widely influential essay (Raymond, 1998) that characterized open development as akin to a *bazaar* style of development, in contrast to the *cathedral* style of development classically described in Fred Brooks' famed *The Mythical Man Month* (twentieth anniversary edition in 1995). We will describe Fetchmail's development in some detail because of its pedagogical significance.

We conclude the chapter with a variety of other important Internet-related open applications. A number of these are free software products that have been commercialized using the so-called dual licensing model. These are worth understanding, first of all because licensing issues are important in open development, and secondly because there is an enduring need for viable business strategies that let creators commercially benefit from open software. The first of these dual licensed projects that we will consider is the MySQL database system. MySQL is prominent as the *M* in the *LAMP* Web architecture, where it defines the backend database of a three-tier environment whose other components are Linux, Apache, Perl, PHP, and Python. Linux is considered in Chapter 3. Perl and PHP are considered here. We describe the influential role of Perl and its widely used open source module collection CPAN, as well as the server-side scripting language PHP that has its own rather interesting model for commercialization. We also briefly consider Berkeley DB and Sendmail (which serves a substantial portion of all Internet sites). Both of these are dual licensed free softwares. Additional business models for free software are examined in Chapter 7. The peer-to-peer Internet utility BitTorrent is a more recent open source creation that exploits the interconnectedness of the Internet network in a novel way and is intellectually intriguing to understand. BitTorrent has, in a few short years, come to dominate the market for transferring popular, large files over the Internet. We complete the chapter with a brief look at the fundamental BIND utility that underlies the domain name system for the Internet, which makes symbolic Web names possible. The tale of BIND represents a story with an unexpected and ironic business outcome.

2.1 The WWW and the Apache Web Server

The story of the Apache Web server is a classic tale of open development. It has its roots in the fundamental ideas for the WWW conceived and preliminarily implemented by Tim Berners-Lee at a European research laboratory. Soon afterward, these applications were taken up by students at an American university, where Berners-Lee's Web browser and server were dramatically improved upon and extended as the NCSA Web server and the Mosaic browser. The NCSA server project would in turn be adopted and its design greatly revised by a new distributed development team. The resulting Apache server's entry into the marketplace was rapid and enduring.

2.1.1 WWW Development at CERN

We begin by highlighting the origins of the Web revolution. The idea for the WWW was originated by physicist Berners-Lee at the CERN physics laboratory in Switzerland when he proposed the creation of a global hypertext system in 1989. The idea for such a system had been germinating in Berners-Lee's mind for almost a decade and he had even made a personal prototype of it in the early 1980s. His proposal was to allow networked access to distributed documents, including the use of hyperlinks. As an MIT Web page on the inventor says,

> Berners-Lee's vision was to create a comprehensive collection of information in word, sound and image, each discretely identified by UDIs and interconnected by hypertext links, and to use the Internet to provide universal access to that collection of information (http://web.mit.edu/invent/iow/berners-lee.html).

Berners-Lee implemented the first Web server and a text-oriented Web browser and made it available on the Web in 1991 for the NeXT operating system. In fact, he not only developed the server and browser, but also invented HTTP, HTML, and the initial URI version of what would later become URLs (uniform resource locators). His HTTP protocol was designed to retrieve HTML documents over a network, especially via hyperlinks. He designed HTML for his project by creating a simplified version of an SGML DTD he used at CERN, which had been intended for designing documentation. He introduced a new hyperlink anchor tag <a> that would allow distributed access to documents and be central to the WWW paradigm (Berglund et al., 2004). Berners-Lee kept his prototype implementations simple and widely publicized his ideas on the www-talk mailing list started at CERN in 1991. He named his browser *WorldWideWeb* and called his Web server *httpd* (Berners-Lee, 2006). The server ran as a Unix background process (or daemon), continually waiting for incoming HTTP requests which it would handle.

At about the same point in time, Berners-Lee became familiar with the free software movement. Indeed, the Free Software Foundation's Richard Stallman gave a talk at CERN in mid-1991. Berners-Lee recognized that the free software community offered the prospect of a plentitude of programmer volunteers who could develop his work further, so he began promoting the development of Web browser software as suitable for projects for university students (Kesan and Shah, 2002)! He had his own programmer gather the software components he had developed into a C library named *libwww*, which became the basis for future Web applications. Berners-Lee's initial inclination was to release the libwww contents under the Free Software Foundation's GPL license. However, there were concerns at the time that corporations would be hesitant to use the Web if they thought they could be subjected to licensing problems, so he decided to release it as public domain instead, which was, in any case, the usual policy at CERN. By the fall of 1992, his suggestions about useful student projects would indeed be taken up at the University of Illinois at Urbana–Champaign. In 1994, Berners-Lee founded and became director of the W3C (World Wide Web Consortium) that develops and maintain standards for the WWW. For further information, see his book on his original design and ultimate objective for the Web (Berners-Lee and Fischetti, 2000).

2.1.2 Web Development at NCSA

The NCSA was one of the hubs for U.S. research on the Internet. It produced major improvements in Berners-Lee's Web server and browser, in the form of the NCSA Web server (which spawned the later Apache Web server) and the

Mosaic Web browser. We will discuss the NCSA server project and its successor, the still preeminent Apache Web server, in this section. The subsequent section will consider the Mosaic browser and its equally famous descendants, which even include Microsoft's own IE.

Like many open source projects, the now pervasive Apache Web server originated in the creativity and drive of youthful computer science students. One of them was Rob McCool, an undergraduate computer science major at the University of Illinois and a system administrator for the NCSA. McCool and his colleague Marc Andreessen at NCSA had become fascinated by the developments at CERN. Andreessen was working on a new Web browser (the Mosaic browser) and thought the CERN server was too "large and cumbersome" (McCool et al., 1999). He asked McCool to take a look at the server code. After doing so, McCool thought he could simplify its implementation and improve its performance relying on his system administration experience. Of course, this kind of response is exactly what Web founder Berners-Lee had hoped for when he had widely advertised and promoted his work. Since Andreessen was developing the new browser, McCool concentrated on developing the server. The result was the much improved NCSA httpd server.

While McCool was developing the improved httpd daemon, Andreessen came up with a uniform way of addressing Web resources based on the URL (Andreessen, 1993). This was a critical development. Up to this point, the Web had been primarily viewed as a system for hypertext-based retrieval. With Andreessen's idea, McCool could develop a standardized way for the Web server and browser to pass data back and forth using extended HTML tags called *forms* in what was later to become the familiar Common Gateway Interface or CGI. As a consequence of this, their extended HTML and HTTP Web protocols "transcended their original conception to become the basis of general interactive, distributed, client-server information systems" (Gaines and Shaw, 1996). The client and server could now engage in a dynamic interaction, with the server interpreting the form inputs from the client and dynamically adapting its responses in a feedback cycle of client-server interactions. Gaines and Shaw (1996) nicely describe this innovation as enabling the client to "transmit structured information from the user back to an arbitrary application gatewayed through the server. The server could then process that information and generate an HTML document which it sent back as a reply. This document could itself contain forms for further interaction with the user, thus supporting a sequence of client-server transactions."

In traditional open development style, McCool kept his server project posted on a Web site and encouraged users to improve it by proposing their own modifications. At Andreessen's recommendation, the software was released