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/w̃ilkins/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.