# Open Source's Lessons for Historians

*Nathan L. Ensmenger*
***University of Pennsylvania***

Of all the developments in the recent history of computing, none has attracted such widespread attention as the emergence of the open-source software movement. In part, this is due to the remarkable successes of such open-source projects as Linux, Sendmail, and Apache. Versions of the GNU/Linux operating system are used by 40 percent of large American corporations, 65 percent of the world's Web servers run Apache, and Sendmail manages 80 percent of the world's email. Even traditional commercial vendors such as IBM, Apple, and Novell have jumped on the open-source bandwagon; the Macintosh OS X operating system is based on a BSD derivative, and IBM recently announced a $1 billion commitment to open-source development.[1]

Despite these apparent successes, however, the lessons of the open-source movement are not necessarily those that its proponents might hope or imagine. They suggest more about new methods and questions for historians to grapple with than obvious conclusions about a new "one best way" to manage software development.

### *The various meanings of "openness"*

At its most basic, open source refers simply to software that is made publicly available as uncompiled source code. Such openness is unusual but hardly unprecedented. There can be compelling reasons for making source code available, even in competitive commercial environments in which intellectual property rights are generally fiercely protected. After all, secrets are just one way of protecting intellectual property; there are others, equally common and effective.

In popular usage, however, the designation of open source is often used to refer to software that is not just open but also free. Free software, in this case, refers to software source code that is both publicly available and free (mostly) of license and copyright restrictions. In general, the only restrictions imposed are designed to assure that it continues to remain unrestricted. Finally, software that is free (of licensing restrictions) is often also made available free of cost. Most often when people talk about open-source software they are referring to projects that are simultaneously open, free, and free of charge.

The appeal and power of open-source software goes beyond these notions of openness and freedom, however. The open-source philosophy is as much about process as it is product. Although advocates argue that open-source methods produce the best software in technical terms (in large part because it can harness the many eyes of a larger community to develop, test, and debug software), these methods are also the best in a larger social, moral, and political sense. As Steven Weber suggests in his excellent book *The Success of Open Source*, "Technical discussions [in the open-source community] on how things should work and should be done are intimately related to beliefs about and reflections on social practices"[2] Open-source projects represent an idealized vision of democracy in action—run by volunteers and organized only on an informal, ad hoc basis; decisions are made by consensus rather than decree. Project participants are independent agents—often widely distributed geographically—who communicate largely through email and newsgroups. There are no scheduled meetings, no rigid job descriptions or organized division of labor, no managerial hierarchies or process controls.

In short, open-source development projects seem to ignore almost all the supposed lessons of modern industrial manufacturing. By radically altering the social and political organization of software development, open source seemingly solves some of the most intractable problems that have plagued the software industry over the last several decades. Theological debates about the intricacies of intellectual property law and licensing schemes aside, the open-source movement is about new ways of managing complex development projects. Increasingly, this applies not only to software; the open-source model has been extended (in theory) to include a wide range of problem domains.

It is easy to see why the open-source movement has attracted so much popular and scholarly attention. It is at once political, social, and technical. By freely giving away valuable intellectual property, open-source developers turn on its head one of the fundamental assumptions that neoclassical economists make about basic human motivation. As seemingly self-organizing communities, they suggest interesting new ways for ethnographers and political scientists to think about the process of self-governance. Also, as part of the anti-Microsoft movement, they represent hope to those who resent large corporate monopolies and the constraints (and expenses) of propriety software systems. For programmers and engineers feeling trapped by cubicles and bureaucracy, open source is an opportunity for creativity, autonomy, and self respect. For everyone else, open source offers up colorful stories full of eccentrics, underdogs, and 20-year-old billionaires. In many ways, the open-source movement has become a kind of Rorschach blot in which everyone sees what they are already looking for.

*continued on p. 102*

**In many ways, open source can only be understood as the most recent chapter in a long-running debate that encompasses the entire history of software development.**

*continued from p. 104*

### *The future of open source's past*

Despite the open-source movement's diverse appeal (or perhaps because of it), the literature on the subject is still immature. Much of it is ahistorical and lacking in context, is written by advocates, and is therefore uncritical or takes too seriously such vague notions as "self organizing" or "gift economies." Let me therefore suggest two potential contributions that the history of computing can make to understanding open-source development lessons and speculate on several ways in which historians could in turn benefit from this discussion.

First of all, much of the literature on open source emphasizes its radical break with the dominant model of closed-source, proprietary, packaged commercial software. Although this model might dominate in the modern PC software market, it is by no means universal, particularly if we are looking across the entire history of electronic computing. In fact, the very notion of software is a historically constructed category. Although the term itself has been around since the 1850s, it was first applied to computing only in the late 1950s. It was not until the 1960s that it became widely adopted, and even then, it was often used to refer to systems software (later operating systems) rather than applications. What little commercial software that was available from outside vendors (as opposed to being developed in-house) was generally bundled with hardware and was therefore not purchased as such.

In the first decades of electronic computing, even software developed internally was often freely distributed (in both meanings of the word) by user groups such as SHARE (IBM) and the Univac Scientific Exchange (USE).[3] It was not until the late 1960s, with the IBM unbundling decision, that software developed into a packaged product. In the mainframe world, purchasing software was often as much about consulting services and support as it was the actual application code. The emergence of a market for closed-source, packaged software was not an obvious or inevitable development; like most other developments in the history of computing, it is the result of complex historical processes playing out in a larger social, technical, legal, and economic environment.[4] In this light, open-source software is part of a continuum of possible configurations of technology, markets, and intellectual property regimes, rather than a radical departure from some natural arrangement.

Second, what is often seen as the open-source movement's most distinctive characteristic—its seamless blending of technical, social, and political agendas—has been a central feature of the history of software since at least the 1950s. Developing good software is notoriously difficult, and the central themes of the persistent "software crisis" that has plagued the industry since the early 1960s are as much social as they are technical:

- the problem of managing complexity and communications within large organizations;
- the difficulties inherent in negotiating between business and technical objectives; and
- the rising costs associated with user support, training, documentation, and maintenance.

Open-source advocates are not the first to recognize the social and organization dimensions of this crisis. A widely cited 1968 article by Melvin Conway argued that a system's architecture reflects the communications structure of the organization that produced it.[5] Frederick Brooks' 1975 classic book *The Mythical Man-Month* described the failure of the OS/360 software development project as one of organization and communication.[6] The solutions to these problems proposed in this earlier period might not have been so explicitly social and political as open source, but every effort to establish a software component's factory, a chief programmer team, or a new software engineering methodology carried with it deeply embedded assumptions about organizational structure, professional status and authority, and individual autonomy.

Are software developers skilled professionals or factory laborers? Active participants in the planning and design process, or mere subordinates to managers and analysts? Is computer programming art, science, or routinized industrial work? In many ways, open source can only be understood as the most recent chapter in a long-running debate that encompasses the entire history of software development.

Finally, some lessons from the open-source movement for historians: Pay attention to the social context of software development. If nothing else, the open-source movement has revealed that seemingly technical debates about "the one best way" to manage software development are also debates about organizational and professional politics, status and identity, and moral significance. Software workers are important. They have independent ideas and agendas and often play an active role in shaping their social and technical environment. Technologies reflect organizational structures. Processes are as important as products.

## References and notes

1. S. Weber, *The Success of Open Source*, Harvard Univ. Press, 2004.
2. Ibid., p. 88.
3. A. Akera, "Voluntarism and the Fruits of Collaboration: The IBM User Group," *SHARE Technology & Culture*, vol. 42, no. 3, 2001, pp. 710-736.
4. M. Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry*, MIT Press, 2003.
5. M. Conway, "How Do Committees Invent," *Datamation*, vol. 14, no. 4, 1968, pp. 28-31. This idea has become popularly known as Conway's Law.
6. F. Brooks, *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, 1975.

Readers may contact Nathan Ensmenger at nathanen@sas.upenn.edu.