

The Development of Open Source Software and its Potential Implications for Higher Education

Background Paper Prepared for a Meeting Sponsored by the Mellon and Hewlett Foundations
October 6, 2005

Introduction

This document provides background information about the development and use of open source software in a variety of contexts and attempts to highlight issues most relevant to higher education. It was assembled as preparation for the meeting on October 6, the primary objective of which is to discuss whether the potential of open source software warrants active investment and intervention by leaders in the higher education community – more specifically, whether the creation of some kind of coordinating organization would be beneficial at this time. This is a complex and fast-moving topic, and this paper does not attempt to provide a comprehensive overview. Nor is it intended to advocate for a particular point of view about the use of open source software by higher education institutions, or to make a case for or against creating such an organization. This paper aims merely to establish common understanding to facilitate a productive discussion. Due to the substantial demands on participants' time, the paper is brief. For those of you who are interested in probing further, an appendix is provided with links to articles that we found especially informative or thought-provoking.

We have organized this paper as answers and explanations in response to a series of questions. The order of discussion is roughly what we anticipate will be the order in which these topics will be addressed during the meeting.

What is open source software?

There is a worldwide “movement” underway related to open source software development; consequently the term has acquired many different meanings depending on one's perspective. When we refer to “open source” we mean that the software's source code is: 1) made available for others to use, view, and modify; and 2) that it may be redistributed by anyone for free, without royalties or licensing fees to the software owner. In contrast, proprietary source code from purchased software packages generally is not distributed or made fully accessible to anyone and in many cases users are prohibited from copying or redistributing the compiled software.

The term “open source” is also commonly used to denote a community-based model of software development. This is the decentralized approach taken by well-known and established projects such as Linux and Apache, in which a world-wide community of programmers contribute to the development and ongoing maintenance of the code. There is a wide continuum of motives for those who contribute to open source software. At one end of the spectrum are unpaid volunteers motivated by the intrinsic rewards of solving an interesting problem, by the prestige one can gain within the programming community, and by the desire to create something for one's own use. Second, there are programmers assigned to add locally needed features and contribute to the improvement of open source software by organizations that have made a business decision to adopt it. And third, there are companies that choose to enhance open source software because they sell services related to that software. This third case includes companies like IBM, which develops open source software in a centralized, controlled environment. We propose that for this paper “open source” indicates just that the source code is made freely available and can be shared and modified. Our discussion, therefore, focuses primarily on the software itself and what can be done with it rather than on how the software was developed. These two interpretations of

the term open source are difficult to separate completely and are often conflated, and because of that in the course of this paper we will point out a few commonly held beliefs as to the pros and cons of both the community-based and directed development approaches.

What open source projects are currently underway in higher education?

Over the past ten years software has assumed a critical role in higher education. In the early 1990s students and faculty were just beginning to use email in earnest. Today, software is woven into almost every aspect of academe, from enterprise resource planning systems used to administer universities, to the distributed research efforts faculty routinely engage in, to course management systems and online courses used for teaching, to institutional repositories and publishing tools used to disseminate research. Campus networks must support local and remote users, provide a high level of security, and meet a broader array of software needs than most large corporations. These include campus portals, student portfolios, personal information managers, and peer-to-peer file sharing tailored for the academic community. Sophisticated software applications are also playing an increasingly prominent role in some areas of the sciences and social sciences. To meet these demands, many universities have built up large IT departments, and in some cases develop their own software to meet these specific needs.

Over a dozen open source projects have been launched within the higher education community to address many of these needs. The most mature of these is uPortal, which is a free, sharable portal-enabling software which allows academic institutions to provide abridged and customized versions of their campus Websites. uPortal compares well to commercial alternatives and has been adopted by hundreds of institutions worldwide. (See Appendix B for a more detailed description of this project). Another major project underway is Sakai, a community-based software development effort to design, build and deploy a new Collaboration and Learning Environment (CLE) for higher education. Sakai is led by staff at the University of Michigan, Indiana University, MIT and Stanford. To date, the Sakai Project has put out three major software releases, which are being piloted at around 25 institutions. It has also developed an Educational Partner's Program (SEPP), composed of both educational institutions and commercial partners, which now has 84 member institutions worldwide, each of which has committed to contribute \$5,000 -\$10,000 annually over three years. Although these partners receive benefits such as early information on the direction of the Sakai Project, strategic briefings to help plan for Sakai implementation at partner institutions, and input into the project roadmap, it is generally believed that SEPP participants are contributing to develop and be part of the SAKAI community rather than to receive particular value-added benefits. Other open source projects are described further in the enclosed supplement.

There is a considerable amount of enthusiasm about the Sakai Educational Partners Program (SEPP) at this stage in its development, and other grant-funded and early software projects in the community have looked to it as one possible model in considering their own sustainability plans. Without some attempt at coordination, it is possible that multiple projects would form similar administrative structures, resulting in duplication of costs and the prospect of universities and colleges being solicited for funds by numerous projects with overlapping areas of activity. This meeting was motivated in large part by the desire to think through the implications of this path, and consider whether a more active approach should be taken to coordinating the future support for these projects.

Before addressing this question, we felt that it would be useful to provide some background about the open source movement in general and how it might address the needs of higher education.

Is open source for real, and if so, how is it impacting the software industry?

Open source is increasingly viewed as a viable alternative to proprietary software and is having a substantial impact on the way commercial organizations do business. There are myriad examples of this transformation. Linux has a 28% share of operating systems on enterprise server computers, and Apache web server software is used by 67% of all websites. MySQL has over six million active installations worldwide. Major corporations such as IBM, Hewlett-Packard, Sun Microsystems and Intel have become supporters of Linux and hardware suppliers sell machines loaded with Linux software. IBM, Oracle, SAP, BEA, Veritas and Intel, as well as many smaller firms such as Red Hat, have created programming groups assigned to work on open source software and provide support services. Venture capitalists are funding startups that support open source software, typically following variations on the Red Hat model. These examples indicate that open source software is establishing its credibility among established organizations. There is also enormous interest worldwide among developers. The SourceForge.net website lists over 100,000 open source projects in progress covering almost every kind of software and has more than 1.1 million registered users.

Shirish Netke of Aztec Software writes that “open source is not about free software. It is really about a new business model for software services.” This is an increasingly widely shared opinion about the developing software industry. Several new business models are emerging as a result. How has an approach to developing software that started out as a grass roots, even iconoclastic movement evolved into a credible force to be reckoned with by dominant software producers? In large part, because it seems to offer real advantages to users and addresses some major shortcomings of the traditional proprietary model, both in the initial creation and in the ongoing development and enhancement of software. First, many hold that open source can result in higher quality software. Transparency identifies and fixes bugs faster, increases accountability, and raises awareness of the software quality, even within an organization. Second, open source software can offer greater flexibility for the user. Users can choose to avoid becoming locked in to software provided by a particular vendor, forced upgrades, and having to purchase features they do not need. Vendors who support open source software have an incentive to offer superior service because switching costs for customers are much lower than with proprietary software. Open source code can be customized by users, and improvements can be shared among the user community. Users can choose the features that are important for their own institutions. And open source code is based on open standards that foster integration and interoperability. An indirect benefit for users is that, where a credible open source alternative exists, they may gain leverage in negotiations with vendors and can put pressure on them to provide greater flexibility and transparency.

There are also significant disadvantages that some attribute to the traditional open source model. First, since the software itself is distributed for free, funding may not be available to support all the features users want in a timely fashion; those relying on open source software may need to have the capacity to build required features themselves, or build coalitions with others who have similar needs. Second, most early successful open source software has been developed by technical people for technical people; these projects (particularly those following the community development model) did not tend to invite the user-centric design approach that is desirable for end-user applications, although it must be pointed out that recent new open source applications such as Firefox (an internet browser) appear to be overcoming this hurdle. Finally, there is often no legal entity responsible for indemnifying users, who may be exposed to copyright infringement lawsuits. The fact that an established company such as Lloyd's of London is about to launch an open source insurance product shows that this risk is a cause of concern for many – and also illustrates how mainstream this movement has become.

The question as to whether using open source software saves money has been the subject of many disputed studies. Some open source products such as Apache have created demonstrable savings for universities as well as for industry at large. More generally, however, savings on licensing fees and upgrades must always be weighed against the costs of installation, customization, support, maintenance, and training, which together may be higher for open source code than for proprietary software. These costs, which must be included when calculating the total cost of ownership, are often not anticipated or well documented, and budget processes (especially within academic institutions) are not set up to take them into account. The answer in any particular case depends on the type of software in question and the level of customization required. It also depends on the nature of the institution, and the extent to which it would be engaged in developing customized software regardless of whether an open source or commercial option were available. Indiana University comes to mind in this context. With a very large staff and a history of customized software development, making its code open source would seem only to offer the plus side of potentially increased innovation from a community of developers working on the open source code. Still, while it is possible that open source software developed within academia will generate savings for some institutions, for the purposes of our discussion, it seems prudent to assume that the benefits of using open source software derive primarily from the level of independence and strategic control access to the source code makes possible, rather than from direct cost savings.

The community development model compounds some of these pros and cons. It can be argued that relying on a wide community of volunteer developers can enhance the quality of the software, as many sets of eyes review the code and contribute to a distributed process of improvement that benefits from a diverse set of perspectives and capabilities. Such an approach may reduce local costs of development, as proprietary software vendors must employ large testing and quality assurance groups in addition to programming groups. Obviously, development costs may be further reduced if contributors are truly volunteering their own time. On the other hand, this model can impede decision-making and prioritization, and is less suited to enforcing timelines and ensuring that less interesting tasks are completed. Projects such as Apache have created governance structures and rules to guide their development, but it does not mandate that anyone take responsibility for such tasks. There are clearly trade-offs, and a wide spectrum of possible arrangements, between the extremes of a command and control hierarchy of software development and an environment that “lets a thousand flowers bloom.”

Should higher education develop open source software to meet its own needs?

Champions of higher education open source software projects argue that academic institutions have unique needs that are not well served by commercial vendors because the sector is too small to support a robust competitive market. Higher education accounts for a relatively small share of the \$250 billion IT industry and appears to be reaching a plateau. According to the College Technology Review, technology spending by the higher education sector will be \$5.15 billion in 2005, 4% less than 2004, with spending split almost evenly between administrative purposes and academic areas. This represents just 2% of the industry. There is a real concern that the relatively small size of this market does not attract the level of competition some would like to see in areas of critical interest to higher education institutions such as course management and financial systems. As a result, one of the most often-heard complaints is that available software options do not have the features required or allow for cost-effective customization. To cite an example from the library field, the 3-4 viable options for integrated library systems, all of which are proprietary, are expensive, buggy, slow to improve, do not allow customization, and are generally unpopular with users. There is at present no open source alternative, and one potential way to address this

problem is for the community to develop its own software, and to pressure commercial vendors to make their products more open, more flexible and more easily adaptable.

Another common concern of many administrators these days is that academic institutions are ceding too much control for mission-critical tasks to commercial vendors. The acquisition of PeopleSoft by Oracle in late 2004 is probably the most dramatic example. After spending millions of dollars installing PeopleSoft, many university administrators feared that Oracle would stop supporting PeopleSoft's financial systems in an effort to force their institutions to switch to Oracle products. They were also concerned that the consolidation of the two leading providers of this type of software to higher education would result in higher prices. A second commonly cited example of higher education's dependence on proprietary software is Blackboard. Over the past seven years Blackboard has acquired almost a 50% share of the course management market. Once an organization achieves a high market share in a small market, the barriers to new entrants discourage new competitors from investing capital. Blackboard's power over its customers has been an ongoing source of frustration, as institutions believe they are forced to buy unwanted features, and expensive customization efforts must be repeated when Blackboard announces that it will no longer support an earlier version of its software.

In addition to pointing to the potential concerns about representing a small market and ceding control, open source advocates argue that higher education has proven it can produce high quality and innovative software, because a number of commercial applications prevalent in higher education originated within academic institutions. Blackboard, the company just mentioned, was originally developed at Cornell. Sun Microsystems' original operating system was the Berkeley Standard Distribution for UNIX. Cisco Systems was built on Stanford networking software. These cases lead some to believe that higher education should retain control over the development of these innovations and business models through which they are distributed rather than allowing them to be commercialized by proprietary vendors. They do not prove, however, that the community has the ability to take these innovations and develop them into sustainable products or attract support from enough institutions to pay for their ongoing development. This is one of the reasons why, in addition to the profit motive and economic incentives for individuals, that commercialization of software has traditionally been the domain of companies. The question being asked now is whether the lowered communication and coordination costs made possible through the advance of networking technologies and the development of the Web make an open source approach a viable alternative.

If higher education were to take an active role in coordinating or supporting the development of open source software, it is important to ask whether such an effort is likely to lead to more success than existing approaches. After all, the disadvantages cited above pertaining to open source (lack of established funding sources, lack of strong track record for non-technical end-user applications, and legal exposure) apply equally to projects developed within the academic environment. Moreover, the development of open source software to serve higher education could also discourage commercial capital from being invested in this space, a development that could well impede innovation. In recent years venture capitalists have funded a number of new enterprises developing course management software, but they may not make those investments if higher education is not prepared to pay for commercial products and is instead committed to offering primary support to internally developed projects. That said, venture capitalists have begun to take an interest in open source software firms following variations on the Red Hat model.

The fact that the higher education market is of relatively modest size also raises the question of whether the academic community is large enough to support its own software. Projects like

Linux and Apache have been built by huge global communities, but then uPortal has been developed with minimal resources and now runs on several hundred campuses and is a well-established and valued option. The Sakai project alone concentrated \$6.8 million invested over two years by the Mellon foundation and the four partner universities who developed it. Blackboard, the leading commercial courseware management system provider, set aside \$14 million last year for research and development on its product. With proper coordination and commitment, could the community development model rise to sustain itself at such a scale? It would depend in part on the extent to which these projects could attract contributions from an international community and the extent to which member institutions supported their IT staffs' participation in such a commons-based production process.

The key question may be how important it is for universities and colleges to have strategic control over various parts of their IT infrastructure. Software such as course management systems are central to the teaching mission of a college or university, as are systems to support research and collaboration among colleagues. Once faculty have become accustomed to a particular interface and invested in putting their course materials into a particular system, the switching costs are extremely high. Is it wise for higher education to be exclusively reliant on a commercial solution for such a core function? Administrative systems for managing student information and payroll are less mission-centric but equally critical to a smoothly functioning institution. And then there are the many software applications described above which are used to enhance the IT network. Understanding the potential dependencies inherent in using today's technology is a very important strategic question for leaders of higher education.

What approach should higher education take to securing the future of open source projects?

If it is determined that higher education should develop its own software in at least some of these areas, one comes back to the question of whether the community should take an active role in securing the future of these efforts. At one end of the spectrum, it could take a laissez-faire approach, allowing these projects to proceed independently and sink or swim on their own merits. In theory a market mechanism would be in effect, as only the projects that provide clear value would get supported. There are some models in the open source world, such as the Apache Foundation's incubation process, that have evolved into fairly effective mechanisms for using clear metrics to determine whether a project should receive continued support. There is a risk, however, that decisions about supporting projects might be made on a first-come-first-served or some other suboptimal basis, and that valuable projects could be starved of funding for the wrong reasons. Another risk is that, even with financial support, these projects would lack the administrative framework and infrastructure required for long term sustainability. Also, colleges and universities might be reluctant to adopt open source software, even if it were it clearly superior to commercial options, if it involved high switching costs and no arrangements were in place for long term support and development of the software.

A more active approach to securing the future of these projects could take different forms. The most obvious would be to create a central body to coordinate decision making about allocation of resources, prioritize development efforts, channel funds, and develop partnerships with commercial vendors to provide support and maintenance. A more ambitious model might be creating an organization staffed with programmers to maintain and upgrade software itself in cooperation with the community of individuals and institutions that have developed it. There is a risk that any centralized approach would destroy the very dynamics that successful open source projects have displayed. In the October meeting, a primary objective is to frame the questions that would need to be answered to determine which of these courses should be pursued.

Appendix: Supplementary Reading

Benkler, Yochai. "Coase's Penguin, or Linux and the Nature of the Firm," Yale Law Review, Volume 112: 2002. <http://www.yale.edu/yalelj/112/BenklerWEB.pdf>.

Ferguson, Charles. "How Linux Could Overthrow Microsoft," MIT Technology Review, June 2005. http://www.technologyreview.com/articles/05/06/issue/feature_linux.asp?p=1

Gleason, Bernard. "Open-Source Software Fosters Integration and Stability." The Chronicle of Higher Education, August 1, 2003. <http://chronicle.com/weekly/v49/i47/47b01301.htm>

Jackson, Gregory A. "Open Source Is the Answer. Now What Was the Question?", The Chronicle of Higher Education, September 24, 2004. <http://chronicle.com/weekly/v51/i05/05b01701.htm>

Kirkpatrick, David. "How the Open-Source World Plans to Smack Down Microsoft,; and Oracle, And..." Fortune Magazine, Features/Fortune Innovation: Special Report February 23, 2004, p. 92. Available upon request (this article is not posted on Fortune.com).

Netke, Shirish. "The Real Open Source Revolution" SandHill.com, June 6, 2005. <http://www.sandhill.com/opinion/editorial.php?id=30>

Yanosky, Ron and Zastrocky, Michael. "The Questions You Should Ask," The Chronicle of Higher Education, September 24, 2004. <http://chronicle.com/weekly/v51/i05/05b02101.htm>

Young, Jeffrey. "5 Challenges for Open Source," The Chronicle of Higher Education, September 24, 2004. <http://chronicle.com/weekly/v51/i05/05b00101.htm>

[Open Source 2007: How Did This Happen? *Educause Review* July/August 2004.](#)