Reusing Open-Source Software and Practices: The Impact of Open-Source on Commercial Vendors

Alan W. Brown and Grady Booch

Rational Software,
8383 158th Avenue NE, Redmond WA 98052
abrown@rational.com, egb@rational.com

Abstract. One of the most intriguing ways that commercial developers of software can become more efficient is to reuse not only software but also best practices from the open-source movement. The open-source movement encompasses a wide collection of ideas, knowledge, techniques, and solutions. Commercial software vendors have an opportunity to both learn from the open-source community, as well as leverage that knowledge for the benefit of its commercial clients. This paper looks at a number of the characteristics of the open-source movement, offers a categorization of open-source dimensions, and provides an analysis of the opportunities available to commercial software vendors when applying the lessons from the open-source movement.

Introduction

Open-source represents one of the most interesting and influential trends in the software industry over the past decade. Today, many organizations are looking toward open-source as a way to provide greater flexibility in their development practices, jump-start their development efforts by reusing existing code, and provide access to a much broader market of users [1].

However, what is widely referred to as the "open-source movement" is in reality a plethora of relative independent initiatives representing a variety of technology innovations and approaches. Elements of the movement's best practices include broadening the notion of a project team, frequent release of new software builds, greater collaboration across geographically dispersed teams enabled by the Internet, and creation of publicly available source code for adaptation and reuse. While a few common threads and community voices have emerged (e.g., Richard Stallman, Eric Raymond, Linus Torvalds, and Tim O'Reilly), the "community" remains essentially a collection of different activities galvanized by a recognition of the benefits of broader, more open access to the software development process, and to the results of that process.

A number of these innovations and approaches have a direct impact on commercial software vendors. There are some aspects on which these vendors can build, others that are in direct competition, and some that offer partnering opportunities in the spirit of "coopetition".

In this paper we examine the relationship between commercial software vendors and open-source. While there is a great deal that could be said, here we concentrate on a few of the major threads with an eye toward the business and economic aspects of this relationship. Many more aspects of the technical and legal relationships require further exploration. Additionally, this paper assumes that the reader does not require too much background or motivation for commercial software vendors' interest in open-source.

Many commercial software perspectives could be adopted in this analysis of open-source, each with its own peculiarities and concerns. However, in this paper the authors assume the perspective of one particular class of commercial software vendor – vendors of software development methods and tools. This is the community with which the authors are most familiar, and in many respects this view is representative of the broader set of perspectives. Hence, the term "commercial software vendor" can be interpreted to refer to this narrow definition, although with obvious application in most cases to the broader commercial software vendor community.

Reuse and Open-Source

It can be argued that the open-source approach is the software industry's most successful form of large-scale software reuse. Open-source software offers the most astounding range of reusable assets for any software project. Open-source software is available for virtually all activities, runs on every platform, and can be used in almost every business domain for which software is written. The open-source movement is predicated on reuse, in many cases with licensing that insists that derived products are themselves released into the public domain.

Two kinds of open-source software reuse can be observed. In the first case, the reuse of open-source software is a planned strategy to augment a popular free product, taking advantage of the wide informal network of open-source developers and users and thus minimizing internal development resources. Often these efforts focus around highly popular open-source products such as Linux, Apache, or StarOffice. Similarly, efforts may be specifically designed in this way through industry consortia. An interesting example of this is the group of commercial software development organizations who have committed to using the Eclipse platform as the basis for software development tools.[1] Eclipse provides an open-source tools framework with an extensible plug-in architecture. As a result, commercial tool vendors do not have to re-implement common functions for managing menus and toolbars, file explorers, and drawing surface manipulation. Rather, they can concentrate on extending this framework with their own value-added domain-specific behavior.

In the second case, reuse of open source is more ad hoc. Particularly in terms of the infrastructure of software development, there are many open-source software products that are so pervasive that they have become part of the fabric of most development organizations. Examples of such products include EMACS, the GNU compilers, build tools such as Ant, and scripting languages such as Perl, PHP, and TCL.

---

[1] The Eclipse consortium includes IBM, WebGain, Borland, TogetherSoft, Merant and Rational Software. See www.eclipse.org for details.

We observe that the reuse of open-source software is just as prevalent in the commercial world as it is in the academic world. Many commercial software organizations would simply not be able to function without some open-source software.

What Is Open-Source?

There are many different views on open-source. A great deal of confusion and misunderstanding often results because discussions on open-source and its impact are frequently not grounded in some common context.

However, as with many topics, describing a laundry list of different definitions of open-source and then positing a new one is not particularly fruitful. Rather than attempt that here, in this section we simply characterize some of the main elements of an open-source approach, and identify some of the issues that arise from those characteristics.

In particular, we focus here on two aspects of open-source that are of special interest to commercial software vendors: the open-source business model, and the open-source development process. The first - the open-source business model - is relevant because it allows us to discuss why mainstream software engineers[2] are looking toward open-source and what they hope to achieve from it.[3] The second - the open-source development process - is equally important because any development environment offering tools for a software engineering community must be well-matched to the development process being followed. With the growing influence of open-source initiatives on all aspects of software engineering it is important to understand open-source development as it offers insight into the requirements for any set of services aimed at the software development community.

Definition and Reality

There does exist a common definition for open-source used by the open-source community (see http://www.opensource.org/osd.html) that is succinct and well crafted. However, it is rather narrow in scope, referring most particularly to the kind of licensing used in a released product, and the rights of third parties to be able to extend, customize, and reuse that product.

However, in reality most individuals and organizations claiming some affinity or use of open-source see it far more broadly than a licensing issue. They frequently see it in terms of:

- Releasing (part of) the source code for a product to broaden its usage base, and encouraging a 3rd party developer community around that product. This is the essential tenet of the Open Software Foundation (OSF).

---

[2] We hesitate to talk about "mainstream" in this context as too pejorative a term. But alternatives such as "traditional" or "commercial" do not seem any better!

[3] This really refers to project managers and decision makers, rather that practitioners per se.

- A way to break the stranglehold on software development held by a few large software vendors – or more likely a way to "keep them honest" through the presence of competition
- A philosophy that aims at leveraging the skills of a wider base of talented software engineers over the Internet.
- Not building new systems from scratch, or with only proprietary code.
- An approach to releasing systems more frequently in line with users expectations for greater responsiveness and shorter cycle times associated with doing business at "Internet speed".

For many commercial organizations interested in open-source the issue of licensing and public release of source code is a secondary consideration. In some cases it is considered a necessary evil to achieve the other goals. At best it is considered a means to an end, not an end in itself. This is confirmed, in fact, in a recent survey of Linux developers [2] in which half of the people developing Linux application said that less than 25% of the Linux based applications they developed were being released and licensed as open-source.

This begs the question: what is open-source? Because of this diversity of opinion, it may be useful to consider the following categories in place of the general term "open-source"[4]:

- **Open-software** – release of source code with a product, and use of an open-source license to encourage or ensure further source distribution, licensing, and redistribution.
- **Open-collaboration** – Internet-based communication and synchronization involving discussion groups, virtual meeting rooms, shared access to development assets, and so on.
- **Open-process** – external visibility of development and release processes, including external auctioning of some aspects of a project, with coordination of multiple resources from within and outside the organization.
- **Open-releases** – use of frequent product releases (at least weekly, if not more often) that gives access for developers to the latest enhancements, and allows early testing and hardening of fixes and enhancements.
- **Open-deployment** – targeting new product releases to a platform that consists (mostly) of open-source products (such as Linux, Apache, Samba, Postfix, etc.).
- **Open-environment** – development of systems using tools and utilities that consist (mostly) of open-source products (such as emacs, GCC, CVS, GNATS, etc.).

Of course, many of these ideas are not unique to the open-source movement. For example, the ideas of open-process are fundamental to approaches such as extreme programming (XP), and open-deployment is arguably the key motivation for Sun's Java initiative (which is not currently open-software according to the definition above).

Note also that these categories are frequently seen in some combination. For example, it seems that the Printer Group at HP are primarily motivated by a move toward open-releases and an open-process, but not open-source or open-software as defined above. The development may use proprietary tools and services, and the results of these efforts will still be included in proprietary product releases.

The Open-Source Business Model

If the open-source approach – actually, if *any* approach - is to be considered a viable way to develop commercial software, it is important to understand the business model

---

[4] We are not proposing all these new terms to be used formally. Rather, in discussions and readings we find it valuable to keep this set of "virtual categories" in our head to gain a deeper understanding of priorities and expectations.

that allows this to occur. The basic premise of an open-source approach is that by "giving away" part of the company's intellectual property, you receive the benefits of access to a much larger market. These users then become the source of additions and enhancements to the product to increase its value, and become the target for a range of revenue-generating products and services associated with the product.

The key, then, is in deciding which projects and products make good candidates for an open-source approach. In [3], Brian Behlendorf[5] suggests that open-source can be a successful business model for commercial projects only within a limited scope. In any such project the following consideration must be taken into account:

- *Characteristics as a software platform.* As a base for developing other applications, a platform fulfils a key role. Because a platform performs a strategic role in those solutions, it is frequently undesirable to have the platform controlled by a single organization. Hence, an open-source approach is particularly appropriate for software that can become a key platform for others to build upon. It makes the platform widely available, encourages its use across the widest possible user community, and fosters a rich set of 3rd party solutions using the platform.

- *Impact on your revenue model.* The financial impact of giving away core parts of an existing product as open-source must be carefully considered. The financial argument typically made is that by doing this it makes the software much more widely available, offering a much larger base of users for additional products, courses, training, and consulting. Such economic forecasts must be validated before a company should be willing to bet its business on an open-source model.

- *Identified market need.* Making (part of) a product open source is only effective if the product is found to be useful. Providing source code is not in itself sufficient to displace a well-entrenched commercial product, or a rival open-source product. Before an open-source decision is made a full analysis of its value to others must be made.

- *Emphasis on server-focused services.* Successful open-source software has tended to implement infrastructure or server-side services. That is because server-side solutions tend to be more pervasive and stable than desktop solutions, incremental additions to functionality are more easily managed, and the developers using open-source solutions tend to be more knowledgeable in these domains (due to their engineering backgrounds). Another factor is the dominance of Microsoft technology on the desktop – something Microsoft is trying to repeat on the server-side.

- *Choosing the right open-source license.* Many kinds of open-source license exist. These vary based on who and how extensions to the software can be made, and whether those extensions also become public domain. Care needs to be taken choosing the appropriate license, and selecting the open-source software on which you build.

- *Making sure you gain momentum.* Early momentum is needed to form a vibrant community of developers and users. Open-source does not mean that no resources are needed to make the software successful. Typical tasks to be managed include active use of discussion groups, provision of management and

---

[5] Brian Behlendorf is a leader of the Apache web server project, and founder of Collab.net.

infrastructure of open-source tools and software, frequent communication through a web site, responding to many enquiries and suggestions, and so on. An open-source project splits into different factions, or falls into disrepute, if the original developers fail to respond to suggestions, do not interact with the user community, or fail to incorporate submitted fixes and updates.

- *Using appropriate tools*. A suitable environment for sharing, synchronizing, and managing the open-source is required. Typically, many different issues and requests will be made, and new fixes and updates will occur from people all over the globe. In addition, users will expect to be able to have visibility into the tracking and management process. In open-source projects these are often managed using tools such as CVS and GNATS.

In summary, we see that a successful, economically-viable open-source approach requires a number of rather stringent requirements be met. Candidate projects for an open-source approach must be very carefully examined based on these requirements. An interesting illustration of this business decision being made in practice is found in [5]. Here, the authors reached the decision that making key parts of their web authoring toolkit open-source increased the value of their company many-fold due to the much wider market access it provided. It allowed them to build their product's position as a widely used platform for web development from which they can gain revenue in add-on products and services.

The Open-Source Development Model

The common, public image of an open-source project is that of a chaotic series of development "accidents" that by some miracle of nature results in a software that compiles and executes. The reality, of course, is very different. The open-source movement has evolved toward a well-defined development process geared to the unique requirements placed upon it – coordination of large numbers of people across the Internet bound together by common interests and goals. The main characteristics of this process include:

- *Encouraging a wide community of users, testers, and developers*. Much of the power of an open-source approach arises from the fact that a large community of people are available to use and exercise the software. This contributes to early diagnosis of problems, consolidation on which new features provide value, and helps to encourage development of patches, updates, and enhancements to the software.
- *Communicating with developers frequently and vigorously*. Open-source projects are renowned for their open forums containing lively technical exchanges. Requirements, designs, and work arounds are widely discussed. New ideas must be proposed and vigorously defended to ensure that they are accepted and introduced into the code base. In addition, e-mail exchanges are frequently the basis for much of the project's history and design documentation.
- *Releasing new software often*. Due to the large developer base, fixes and updates are submitted daily, if not even more frequently. In many open-source projects this new code is typically baselined into the released software on a daily basis.

For example, it is not unusual in some open-source projects to have multiple releases of a code base each week. Often, this is controlled by also maintaining a stable build on a longer time cycle (say, monthly). This quick release of updates ensures that users who choose to can immediately take advantage fixes and new functionality. With this approach any bad fixes or ill-conceived additions are quickly found and removed.

- *Applying strong coordination*. To support frequent releases there must be a strong coordination model. Often this is due to tight control exerted by the open-source coordinator. Their responsibility is to maintain design authority over the code, make decisions on the appropriateness of each addition, and to apply proper configuration control to the builds. This is the most important role in the open-source development process. It can be argued that every interesting open-source product has a strong intellectual leader and hard-core developer behind it. Each wields a strong hand in the content of that product, making these development projects more like a benign dictatorship than a true bazaar.

- *Being responsive*. The developers involved in an open-source project are not generally motivated by salary, or career enhancement. They tend to get involved due to their need to solve problems, work with their peers in a collaborative environment, and demonstrate their development prowess.[6] It is important that their ideas are examined, analyzed, and considered for inclusion. This requires those in charge of an open-source project to be intelligent, diligent, and responsive. It is the least that the development community expects.

- *Ensuring visibility into the process*. One of the expectations about an open-source project is that there is some measure of openness in the process that is used for development. That requires a set of tools and services that allow developers to submit requests and changes, view their status, and interact with the open-source coordinators to ensure appropriate decisions have been made. This is most often carried out using mailing lists, discussion forums and Internet-based developer portals. Additionally, for larger projects there are meetings, workshops, and developer conferences.

Each of these characteristics contributes to the open-source development process to create a highly interactive approach. In many ways these projects represent the antithesis of the large, "waterfall"-style projects that in the past dominated mainstream software engineering.

When (and When Not) to Reuse Open-Source Software

The discussion above is useful for highlighting the diversity of perspectives on open-source. However, it does little to illustrate the practical considerations faced by many organizations in the most prevalent form of reuse involving open-source – when reusing open-source software as part of a non-open-source software development project. In such situations there are many important considerations.

---

[6] The motivation and ethics of open-source development is complex topic in its own right, and discussed in detail in [3].

Typically, during the project's conception or early analysis phases there are some aspects of the proposed solution that appear to be candidates for reuse of open-source software. At this point all the traditional reuse techniques for evaluating, adapting, and including the open-source software into the new system are applicable. However, due to the open-source nature of the reused software, there are a number of special considerations that must be taken into account. While the functionality and accessibility of the open-course software may be ideally suited for reuse in this new context, these additional considerations are frequently dominant in the decision process concerning whether the open-source software can be reused.

Based on the authors' practical experiences, we have found two issues to be the most difficult and complex to consider: licensing and security. It can rightly be argued that both of these must be considered in any reuse situation. However, practical experience leads us to conclude that they take on disproportionately more significance when open-source software is involved.

It is interesting to note that these concerns are in-line with other commercial software vendors. The most prominent warnings about the problems of reusing open-source software in a commercial context have come from Microsoft. In a recent speech[7] Craig Mundie, a Microsoft Senior Vice President, stated that:

"The [open-source software] development model ... has inherent security risks and can force intellectual property into the public domain."

This concern has led to Microsoft labeling open-source software as "viral" software, and to warnings that open-source software licensing approach such as the GNU Public License (GPL) has a viral nature that poses a threat to the intellectual property of any organization that derives its products from GPL source. We explore these concerns in more detail below.

Licensing

Open-source software licenses are typically rather straightforward, but their implications on any systems reusing that software can be rather profound. The main concern often hinges on the notion of whether the system reusing the open-source software is considered to be a "derived work". If so, according to many open-source licenses the system reusing the software must also be considered open-source software and subject to the same licensing agreements. This is a basic tenet, for example, in the GNU Public License (GPL).

Commercial software development efforts are at most risk from license violations, and of course large commercial organizations are the most attractive targets for being sued for infringements of any licensing agreements. Even the remote possibility that a large-scale commercial product could be claimed to be risking its commercial viability by reusing open-source software frequently results in most commercial organizations insisting on an "open-source review" before the release of any new commercial product. Indeed, most large commercial software organizations have legal

---

[7] See "Prepared Text of Remarks by Craig Mundie, Microsoft Senior Vice President, The Commercial Software Model, The New York University Stern School of Business, May 3, 2001" - http://www.microsoft.com/presspass/exec/craig/05-03sharedsource.asp.

experts on call who specialize in ensuring the legal ramifications of reuse of open-source software are well understood.

From a legal perspective there seem to be four possible strategies to consider when reusing open-source software:

1. Don't reuse open-source software;
2. Clearly separate (architecturally) the pieces of the system that rely on open-source software from independent layers that leverage them – the independent layers can then be considered proprietary;
3. Hire a good open-source software lawyer;
4. Place all the resulting software in the public domain too!

Security

Recent publicity has highlighted the inherent insecurity of many of the software-based systems that we all rely on everyday. These concerns have increased the scrutiny on how software is developed, and in particular the origins of much of this software. Reuse of open-source software has raised the most concern. The media makes a great deal of the fact there can often be many individual contributors to a single open-source software system. As a result, they ask "if you don't know who developed your software, how do you know what it will do?"

Clearly, there is some substance to this argument. Large software systems involve millions of lines of code developed over an extended period of time by literally hundreds of developers. Verifying that there are no hidden security flaws (whether intended or otherwise) is practically impossible. When part of the system is a piece of reused open-source software, the risk of security problems increases.

Of course everyone does not hold this view. As open-source advocates point out, open-source software such as Linux and Apache is probably the most widely known and used code base in existence. The wide availability of the source code has meant that many thousands of programmers around the world have at some stage looked at and examined parts of the software. In contrast, large commercial systems are rarely released as source code. Therefore, as a consumer you have no ability to find out what that system really does.

Regardless of many company's positions on this debate, the possibility of security issues by reusing open-source software has itself been sufficient to ensure that many companies do not reuse open-source software in their systems. Instead, they rely on the support provided by commercial systems.

From a security perspective there seem to be three strategies you can take when reusing open-source software:

1. Don't reuse open source software;
2. Only reuse open-source software that has been through extensive internal code reviews typical of all software you develop;
3. Foster a strong relationship and understanding of the open-source software community, closely follow the newsgroups for open-source software that you reuse, and get involved in the development of that open-source software!

Taking Advantage of Open-Source in a Commercial Setting

Commercial software is rarely released as open-source, simply because there is no sustainable economic business case for such a practice. Typically in commercial settings there is no intention to release the developed software into the public domain using one of the open-source licenses. In fact, in many cases it would be impossible to do so given that at its core many such solutions aggregate, integrate, and extend commercially acquired tools and services.

However, commercial software vendors have been significantly influenced by open-source initiatives, and leverage open-source products in a number of areas. This can be seen through inclusion of open-source software, in the targeting of open-source technologies for deployment, in support for the creation and management of interactive open communities, and in the move to supporting open styles of collaborative software engineering.

Here we review how commercial software organizations can best take advantage of the open-source movement. In the discussion below we focus on a specific class of commercial software vendors – the commercial software tools vendors, as represented by companies such as Rational Software.

Targeting an Open-Source Deployment Platform

Almost all organizations have standardized on (some flavor of) Microsoft Windows as the platform for the desktop, running client applications such as Microsoft Office Suite. However, many commercial organizations are now looking to deploy server-side solutions to a target platform containing a number of open-source products. Typically, this refers to the Linux operating system and Apache web server. Much of the demand for server-side open-source deployment is due to the perceived lack of reliability and speed offered by Microsoft, and the failure to offer low priced commercial server-side solutions by Microsoft's main rivals. As a consequence, commercial organizations are attracted by the low cost of setting up powerful server farms of Intel-based machines running open-source software.

However, there are many elements required to achieve success in the deployment of solutions to an open-source platform. Currently there appear to be three major barriers to success:

- *Few development tools*. While the open-source community provides a number of code-centric development tools, in general there are few robust, team-based tools supporting the broader software life-cycle. A wider range of tools for deploying applications to open-source platform is required.
- *Lack of practical advice on open-source deployment in a commercial setting*. There is little available knowledge on how to design, implement, and manage server farms constructed from groups of open-source products. In this context commercial organizations require a great deal of guidance and advice on many aspects of the software development process, models and "how-to" guides defining common architectural solutions, and heuristics and metrics on performance.

- *Scarcity of available skills*. There are few people currently with the skills to develop robust open-source deployment platforms for use in a commercial setting. To be successful requires not only greater training and education, it also requires access to a broader community of open-source expertise, and the opportunity to share knowledge and experiences with other organizations following a similar approach.

Commercial software vendors are in an excellent position to provide solutions to each of these three barriers.

First, commercial software vendors offer a range of best-of-breed development services covering the complete life-cycle. The tools offered by commercial software vendors are far in advance of anything currently available in the open-source community.

Second, the commercial software vendors approach to content management is to allow practical advice and information to be made available to the right people at the appropriate time in the development process. It would be an excellent basis on which to create a body of knowledge on deployment to an open-source platform in a commercial setting.

Third, the commercial software vendors' community activities and infrastructure support communication, and asset sharing among organizations using the commercial software vendors services. Many of these mechanisms could be focused on providing access to a range of expertise to commercial organizations targeting open-source deployment.

However, to support these communities effectively requires commercial software vendors to take a number of initiatives directly aimed at making progress in this area.

Measuring Up to the Open-Source Bar

There are already a number of organizations offering hosted development and collaboration services to the open-source community. The three main contenders here are Collab.net, OpenAvenue (which includes SourceXchange), and SourceForge.[8] Other sites not directly targeting the open-source community also offer free hosted services frequently used by open-source projects (e.g., egroups). Their primary services are in the areas of collaboration and discussion forums, code-centric tools, and project hosting.

To a large extent it can be said that these organizations have already "set the bar" with respect to a minimum set of services expected by organizations today as collaborative tool support for distributed teams of developers. The sites offer the standard against which other commercial software vendors will be measured. As a result, it is essential that commercial software vendors understand the services they offer, and can articulate their offering in relation to them. In this sense, the open-

---

[8] There are also many sites focused on information exchange (e.g., slashdot as well as platform-specific portals such as MSDN and developerWorks).

source offerings already available provide an important "touchstone" that the commercial software vendors can use to measure their services and capabilities.

Leveraging the Open-Source Community

One of the most intriguing questions concerns how commercial software vendors should leverage the open-source community. In particular, the open-source community represents a large body of software developers who are themselves potential clients for the commercial software vendors' offerings. Furthermore, they represent a significant set of influencers, trendsetters, and thought leaders in the software engineering community. There appear to be at least 3 possible opportunities for commercial software vendors to leverage the open-source community:

- *Target the open-source community as clients of commercial software products*. It is possible to consider integrating open-source artifacts to provide access to commercial software vendors services, and to encourage interaction between the open-source community and the commercial community. This has many attractions for both commercial software vendors and its clients. However, it must be balanced with the knowledge that there is little chance of direct revenue from this strategy because commercial software vendors will be in direct competition with the freely available open-source solutions. This may do more to expose commercial software vendors' weaknesses and gaps, rather than highlight its strengths.
- *Actively attract the open-source community to visit, take part in discussions, and share expertise*. While not integrating open-source projects, it is still desirable to attract the open-source community to commercial software vendor communities. They offer a perspective on software development that is essential to today's commercial organizations. Additionally, they may provide a marketplace for commercial organizations to obtain consulting and development resources. Commercial software vendors could position themselves as the place where "open-source meets the commercial market" for the benefit of both parties.
- *Provide a synthesis of open-source knowledge, technology, and techniques for use in a commercial setting*. A valuable service that commercial software vendors could offer would be to offer a synthesis of the open-source community for commercial software developers. There is so much knowledge and assets available for use, the commercial world has a great deal of trouble deciding what may be useful. Commercial software vendors could be sufficiently "plugged-in" to the open-source community to offer the best advice, knowledge, techniques, and solutions. This would be of tremendous value to the commercial community. It gives commercial software vendors a clear role, and does not directly compete with other open-source offerings.

Summary

As a result of the open-source movement there is now a great deal of reusable software available in the public domain. This offers significant functionality that commercial software vendors can use in their software projects. Open-source approaches to software development have illustrated that complex, mission critical software can be developed by distributed teams of developers sharing a common goal.

Commercial software vendors have an opportunity to both learn from the open-source community as well as leverage that knowledge for the benefit of its commercial clients. Nonetheless, the open-source movement is a diverse collection of ideas, knowledge, techniques, and solutions. As a result, it is far from clear how these approaches should be applied to commercial software engineering. This paper has looked at many of the dimensions of the open-source movement, and provided an analysis of the different opportunities available to commercial software vendors.

References and Notes

1. It can be argued that the open-source community has produced really only two essential products[9] -- Apache (undeniably the most popular web server) and Linux – although both are essentially reincarnations of prior systems. Both are also somewhat products of their times: Apache filled a hole in the then emerging Web, at a time no platform vendor really knew how to step in, and Linux filled a hole in the fragmented Unix market, colored by the community's general anger against Microsoft.
2. Evans Marketing Services, "Linux Developers Survey", Volume 1, March 2000. This provides a number of interesting statistics on the current practices of open-source developers.
3. Eric Raymond, "The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary", O'Reilly Press, 2000.
   Raymond positions open-source as a logical evolution of previous development practices. In particular, argues that there is a strong sense of control, ownership, and process that is common to open-source projects.
4. Chris DiBona et al. (eds.), "Open Sources: Voices from The Open Source Revolution", O' Reilly Press, 1999.
   A selection of papers from the most prominent people in the open-source movement.
5. Paul Everitt, "How We Reached the Open Source Business Decision", Zope, http://www.zope.com/Members/paul/BusinessDecision, 1999.
   An enlightening case study on the business decision to go open-source, driven by the need to create a platform for web development.
6. David Brauer, "Toyota to save $3M a year with the help of Linux", Enterprise Linux Today, http://eltoday.com, October 24, 2000.
   The reason that Toyota chose Linux and Apache to connect their 1200 dealerships. Focuses

---

[9] Of course, there is Emacs, sendmail and all sorts of related products, but most of these are the products of a single person with contributions from various Darwinian-selected contributors. There are also products such as Sun's open source office product, but those did not rise organically from the primordial seas of the open source community, but rather were seeded by their patrons to serve some larger business goal.

on the small footprint, connectivity, and the deal between Redhat and Gateway to produce Linux boxes.

7. David Brauer, "Stocktalklive.com – A Business Case for Linux", Enterprise Linux Today, http://eltoday.com, July 17, 2000.
    A case study of stocktalklive.com and their use of Linux.

8. Alan MacCormack and M. Iansiti, "Developing Products on Internet Time", *Harvard Business Review*, 75th Anniversary Edition, Sep-Oct 1997.
    A look at the changing business practices in the software community due to the need to create systems more quickly and with greater flexibility. An extract from McCormack's Ph.D. thesis on this topic.

9. George Lawton, "Opening the Gates to a New Era", ADT Magazine, October 2000.
    A general discussion of the move to open-source. It contains a couple of interesting short case studies, and some industry opinions.

10. Ming-Wei Wu and Ying-Dar Lin, "Open Source Software Development: An Introduction", IEEE Computer, Vol. 34 No. 6, June 2001.
    A useful overview of open source initiatives, concentrating on the various kinds of licenses.

11. Haiqing Wang and Chen Wang, "Open Source Software Adoption: A Status Report", IEEE Computer, Vol. 18 No. 2, March/April 2001.
    A review of various open source software available, and a framework for evaluating the appropriateness of various open source software systems.s