The Business Prospects of Open Source

Dabos Halloran Luo
17-910 Project

May 7, 2002

"Software is largely a service industry operating under the persistent but unfounded delusion that it is a manufacturing industry." — Eric S. Raymond [38]

Abstract

Open source software is not the end of the commercial software industry but it is clearly changing some areas of the industry's landscape. In this paper we examine the open source development method as it exists in practice today from a technical and social context. We find similar licenses, leadership by meritocracy, and tool-based collaboration across eleven projects and three development portals surveyed. We propose a "walled server" model to describe open source project information flow and control. We further examine the social context of open source communication and intragroup conflict. We describe five business models observed in today's open source industry and evaluate when a business should choose to use open source methods over other approaches. We conclude by proposing four areas for future research: open source patronage, competition between open source and proprietary software, open source use as Off-The-Shelf (OTS) software, and transplanting the open source software development method.

1 Introduction

The widely publicized success of open source software has created an explosion of business interest in open source software. A number of open source products, such as the Apache web server or Linux operating system, have notably consolidated their competitive position during the last few years. In particular, the April 2002 Netcraft survey [32] of 37.5 million Internet domains reported Apache to hold a dominant position with 56.38 percent of the respondents using this web server. Turning the attention to the operating system market, International Data Corporation in 2000 estimated the number of Linux users between 7 and 16 million users worldwide, with a 200 percent annual growth rate. Open source software represents an important challenge for traditional models of software development based on proprietary control over both the software code and the process of development. In fact, experts in the field have held up open source software as a potential challenge to Microsoft's long-standing dominance in the PC operating system market (e.g., expert witnesses in the antitrust case against Microsoft, the Halloween Papers [14][1] ; and Rosenberg [39]).

Unfortunately, open source software is surrounded by inaccurate and extreme characterizations. At one extreme it is a titanic struggle of good, personified by Linux creator Linus Torvalds, verses borg-like evil, personified by Microsoft founder Bill Gates [13]. At the other extreme it is an unAmerican anti-business assault on a legitimate software industry [7] led by freaky neo-communists, often personified by Richard

[1]The term "Halloween Papers" is used to refer to two internal and confidential Microsoft memos describing the open source phenomenon and analyzing possible strategic responses to combat this latest market threat. The memos were slipped to Eric Raymond, one of the most prominent members of the Open Source Initiative, by a source who remains nameless. Once the memos and Raymond's comments on them were released to the press, Microsoft publicly acknowledged their authenticity.

Stallman. These glib characterizations fail, miserably, in their analysis of the true impact and implications of open source software on business. When should a business use open source software? Should a business develop software using open source methods? What are the claimed benefits of open source software and open source software development methods? Are these claims true? Finding real unbiased answers to these questions, we believe, is crucial for any business that uses or develops software today.

In this paper we examine the open source development method as it exists in practice today from a technical and social context. We propose several business models observed in today's open source industry and evaluate when a business should choose to use open source methods over other approaches. We conclude by proposing four areas for future research.

2 The open source method

In this section we examine the strong ties between open source and software licenses and then describe the attributes of the open source software development method as observed in practice. We conclude the section with a summary of the key attributes of the open source method.

2.1 The software license

Open source software constrains the license under which it is distributed. In fact, most definitions and descriptions of open source software focus on describing attributes required of the software license rather than the actual software development method. This license-oriented approach provides little insight into the actual methods and practices used to develop open source software but is critical to understanding the practical limits open source software places upon any company using or developing open source software.

Today, the most widely accepted definition of open source is the Open Source Definition (OSD) maintained by the Open Source Initiative (OSI) [9]. The required attributes of OSD conformance, quoted from [36], are:

- *Free Redistribution.* The license may not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license may not require a royalty or other fee for such sale.

- *Source Code.* The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost—preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

- *Derived Works.* The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

- *Integrity of The Author's Source Code.* The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

- *No Discrimination Against Persons or Groups.* The license must not discriminate against any person or group of persons.

- *No Discrimination Against Fields of Endeavor.* The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.
- *Distribution of License.* The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.
- *License Must Not Be Specific to a Product.* The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.
- *License Must Not Contaminate Other Software.* The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

A software license must conform to every OSD attribute listed above to be considered open source. The legal terms of open source licenses focus on preserving free redistribution of the software, source code access, and allowing derived works. This focus is quite different from common commercial software licenses which focus on limiting software redistribution and preserving intellectual property rights.

There is one further open software license issue we need to clarify before continuing, the issue of *copyleft*. A significant portion of open source development falls under the GNU General Public License (GPL) [5] which is the flagship license for the Free Software Foundation, founded and led by Richard Stallman. The Free Software Foundation maintains another definition of open source called the Free Software Definition. Software developed under the Free Software Definition is commonly called *free software* rather than just open source. Free software uses *copyleft* as an enforcement vehicle. From the Free Software Definition: "copyleft (very simply stated) is the rule that when redistributing the program, you cannot add restrictions to deny other people the central freedoms." [4] The addition of copyleft changes the rules that derived works must follow. Specifically, free software generally prohibits proprietary derived works. An example best illustrates this difference. The Linux operating system kernel is licensed using the GPL. Several companies redistribute the Linux kernel (e.g., Red Hat, SuSE, Mandrake) but all are required to distribute it with source code and they cannot restrict further redistribution of their work. The use of copyleft is often called (by its detractors) *viral*—once it is in your software you are infected and are now free software forever [18].

Non-copyleft open source licenses are common today, examples include the Apache software license and the Berkeley Software Distribution (BSD) license. These licenses allow derived works that are not OSD compliant. A company is free to create a proprietary version of Apache and license it under commercial terms. As noted above, because of copyleft, no company can do the same for the Linux kernel.

Armed with an understanding of the key license issues that underpin open source, we now describe the open source software development method.

2.2 The method in practice

The modern open source development method emerged from the development of the Linux operating system kernel [41]. The public and commercial success of Linux and the development method used by Torvalds, and later documented and advocated by Raymond in [37], are the roots of the method used by most open source projects today. Even open source projects that predate Linux, specifically Free Software Foundation (or GNU) projects, have transitioned to the modern open source development method.

Table 1: Open source project size, collaboration tool use, and description (March 2002)

	Size & Composition		Collaboration	
Name	kSLOC	Language	Tools Used	Description (web site)
Apache HTTPD Server	77	*Total*	CVS, Ezmlm, GNATS, ViewCVS	A web server (apache.org)
	70	C (92%)		
	5	sh (7%)		
GNOME	1,338	*Total*	CVS, Bonsai, Bugzilla, IRC, LXR, Mailman	A Unix/Linux desktop environment (www.gnome.org)
	1,207	C (90%)		
	86	sh (6%)		
GNU Compiler Collection (gcc)	1,182	*Total*	CVS, DejaGnu, Ezmlm, GNATS, MHonArc	A programming language compiler (www.gnu.org/software/gcc)
	884	C (75%)		
	163	C++ (14%)		
	65	Java (6%)		
Jakarta Tomcat	59	*Total*	CVS, Ezmlm, Bugzilla, ViewCVS	A Java (Servlet/JSP) web server (jakarta.apache.org/tomcat)
	44	Java (75%)		
	14	C (24%)		
KDE	2,050	*Total*	CVS, Code/CVS Web, DebBugs, LXR, Mailman	A Unix/Linux desktop environment (www.kde.org)
	1,717	C++ (84%)		
	228	C (11%)		
Linux Kernel	2,570	*Total*	CVS, CVSWeb, LXR, Mailman, Majordomo	An operating system (www.kernel.org)
	2,415	C (94%)		
	146	asm (6%)		
Mozilla	2,170	*Total*	CVS, Bonsai, Bugzilla, IRC, LXR, Tinderbox	A web browser (www.mozilla.org)
	1,358	C++ (62%)		
	749	C (35%)		
NetBeans	758	*Total*	CVS, Bugzilla, ViewCVS	A Java programming environment (www.netbeans.org)
	753	Java (99%)		
Perl	313	*Total*	CVS, MHonArc, Perlbug, Request Tracker,	A programming language (www.perl.org)
	161	Perl (51%)		
	123	C (39%)		
Python	384	*Total*	CVS, CVSWeb, Mailman	A programming language (www.python.org)
	192	Python (50%)		
	185	C (48%)		
XFree86	1,943	*Total*	CVS, Mailman, ViewCVS	A Unix/Linux windowing system (www.xfree86.org)
	1,833	C (94%)		
	36	C++ (2%)		

Table 2: Open source development Internet web portals (April 2002)

Open Source Web Portal Name	Hosted Projects	Registered Developers	Description (web site)
Sourceforge.net	39,248	415,879	A popular open source development portal (sourceforge.net)
Savannah at GNU	770	6,360	Free Software Foundation development portal (savannah.gnu.org)
Tigris.org	43	unknown	A web portal focused on software engineering tool development (www.tigris.org)

To understand and characterize this method we surveyed several successful open source projects. We examined the publicly visible portions of these projects with a focus on the software development and collaboration approaches used. A summary of results for eleven of the projects surveyed is presented in Table 1. The source lines-of-code (SLOC) counts for the predominate languages are shown in the table in order to provide a rough gauge of code size and complexity. The projects all utilized web portals to encapsulate collaboration tools, and these tools are inventoried in the table (these tools, as well as their significance, will be described below when we discuss boot-strapping). In addition to the projects listed in Table 1, we examined several open source development portals, a summary of which is presented in Table 2. Table 2 presents the number of open source software projects hosted at each of the listed web portals and the number of registered users (registered users may or may not be active developers). All the projects listed in Table 1 use custom web portals except the Python project which is one of the many projects hosted on Sourceforge.net.

The open source projects we surveyed are medium to large in size. The Apache HTTPD project at 77k SLOC was the smallest and the Linux kernel at 2,570k SLOC was the largest. The majority of the projects examined were implemented with the C or C++ programming language. The C and C++ languages are strongly tied to Unix (the operating system of choice for open source and which the Linux operating system mirrors) and have been widely available to programmers at little or no cost. Java also showed up in three of the projects we examined.

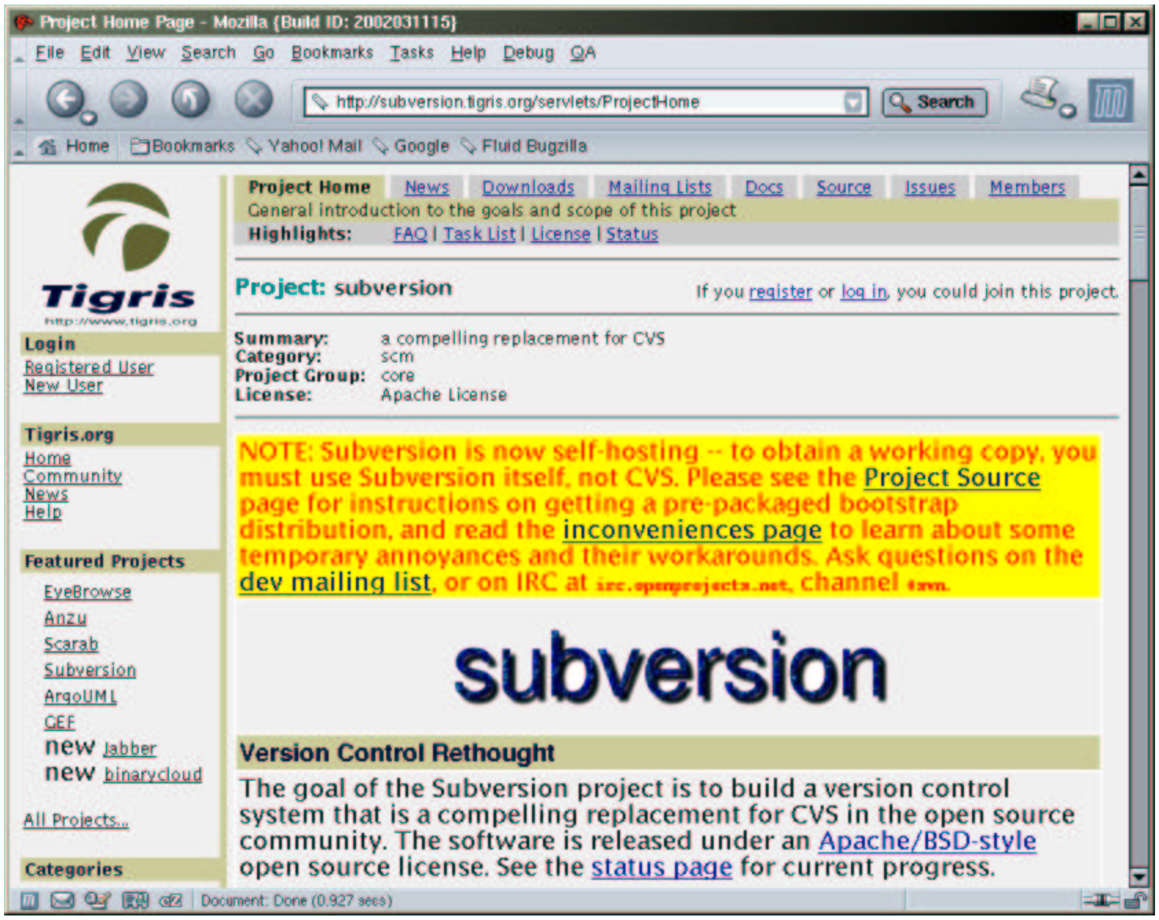
Observed open source practice is surprisingly uniform. Differences exist in the detailed processes and tools used by projects but similarities across projects are more striking than differences. Below we describe some of the attributes the open source method, including communication, leadership and project management, process and the walled server, boot-strapping, incrementality and gentle slope, and evolutionary focus.[2]

2.2.1 Communication

A ubiquitous, almost defining, trait of open source practice is that *tool mediation is the norm.* This enables leaders to shift the burden of policy enforcement from people to tools. Tools support authentication, regulation of commit privileges, audit and notification, and other policy-related functions.

One of the reasons why tool mediation is possible to this relatively unusual extent is that developers are rarely co-located. A consequence is that developer communication is almost always mediated through the project server, creating an *ad libitum* organizational memory. Indeed, projects with significant co-located groups, such as Mozilla and NetBeans, must take explicit measures to reinforce the social norm of computer-mediated developer communication. The textual organizational memory is semi-structured, in the sense that

[2]Portions of the open source method attribute description adopted from *High Quality and Open Source Software Practices* by T.J. Halloran and William L. Scherlis [20].


Project Home Page - Mozilla {Build ID: 2002031115}
File Edit View Search Go Bookmarks Tasks Help Debug QA
http://subversion.tigris.org/servlets/ProjectHome
Search
Home Bookmarks Yahoo! Mail Google Fluid Bugzilla
Tigris
http://www.tigris.org
Login
Registered User
New User
Tigris.org
Home
Community
News
Help
Featured Projects
EyeBrowse
Anzu
Scarab
Subversion
ArgoUML
GEF
new Jabber
new binarycloud
All Projects...
Categories
Project Home News Downloads Mailing Lists Docs Source Issues Members
Highlights: FAQ | Task List | License | Status
subversion
Document: Done (0.927 secs)

General introduction to the goals and scope of this project

Project: subversion

If you register or log in, you could join this project.

Summary:	a compelling replacement for CVS
Category:	scm
Project Group:	core
License:	Apache License

NOTE: Subversion is now self-hosting -- to obtain a working copy, you must use Subversion itself, not CVS. Please see the Project Source page for instructions on getting a pre-packaged bootstrap distribution, and read the inconveniences page to learn about some temporary annoyances and their workarounds. Ask questions on the dev mailing list, or on IRC at irc.openprojects.net, channel #svn.

Version Control Rethought

The goal of the Subversion project is to build a version control system that is a compelling replacement for CVS in the open source community. The software is released under an Apache/BSD-style open source license. See the status page for current progress.

Figure 1: A typical open source project web portal at Tigris.org

the discussions are situated in topic- or issue-specific mailing lists. While the conversation is focused on issue resolution, it also (at negligible marginal cost) records rationale for later use.

To be more concrete, a typical open source project web portal is shown in Figure 1. This screen shows the main web page for the Subversion project (a project dedicated to creating a new source code configuration management system, but the details of the Subversion project are not relevant to our discussion) which is hosted by the Tigris.org web portal. The links at the top of the Subversion web page: *Project Home*, *News*, *Downloads*, *Mailing Lists*, *Source*, *Issues*, and *Members* are found, in principal, on all open source web portals we examined. The *Project Home* describes the functionality of the project's software and introduces the project's community. The *News* page electronically documents major software releases, events, and other items of widespread project interest. The *Downloads* page allows users to quickly download project software and documentation. The *Mailing Lists* page allows public subscription to one or more of the project specific mailing lists. The number of mailing lists varies by the size and focus of the project. The GNU Compiler Collection project uses 19 mailing lists while the KDE project uses 64 mailing lists. Mailing lists are the *primary* communication vehicle for any open source project and every project maintains at least three key mailing lists: an announcement list for project news, a user list to allow user-to-user contact and help, and a developer focused list to allow project management, design, and general discussion between project programmers. All mailing lists are generally open to the public. Private or limited subscription lists are generally frowned upon.[3] The *Source* page describes how to directly access the project's source code via the source code control system used by the developers. The public is allowed a direct, but read-only, interface to the live source code control system for the project. This level of access, which is nearly never available in commercial software, is critical to recruiting new programmers. New programmers must be able to look at where the state of source code now, not what its state was two months ago when the last stable release was made. The *Issues* (also commonly called "bugs") page provides a web based system to report and track software defects and requests for functionality enhancements (it also has a role in project management which will be discussed below). The *Members* page lists the developers contributing to the project and how to contact them.

2.3 Leadership and project management

Open source projects contrast sharply with traditional industry software development methods in the areas of leadership and project management.

Leadership in an open source project is based largely upon technical contribution and achievement, thus open source projects are managed by a *meritocracy*. The traditional role of executive management to align software development efforts with business objectives, at least at the surface, is notably absent. The goals and objectives for an open source project are set by the leaders of the project's meritocracy and also by what the project's programmers are willing and able to accomplish. Open source projects, to remain viable, need to continuously recruit programmers willing to work on the project's goals and objectives. Hence, open source development has a dimension absent from traditional software development methods—recruitment of new talent to work on the project.

Companies working to align the objectives of an open source project with their own business objectives must work through the indirect means of *patronage*. Patronage of an open source project occurs when a company donates programmers, computers, Internet bandwidth, or direct funding. The most famous instance of corporate open source patronage was the widely publicized creation of the Mozilla browser project by Netscape (now AOL) in March of 1998 [16]. A more recent example is IBM's commitment of

[3]An example of an exception to the public mailing list policy is the private mailing list hosted by the Apache web server (HTTPD) project for reporting serious software security problems directly to senior Apache developers, use of this list is encouraged due to potential criminal use of this information (e.g., disruption of commercial web sites) [11].

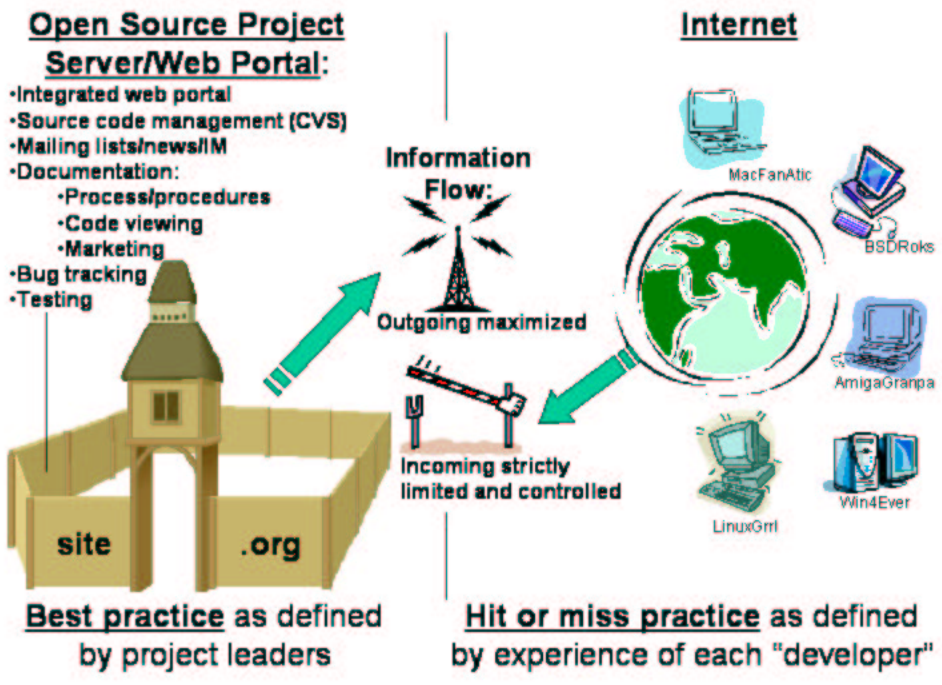


Figure 2: The walled open source project server model

$1 billion toward Linux development and $40 million to create and support the Eclipse project [23]. How much corporate influence is possible and what benefits and drawbacks it brings to an open source project are not well understood.

Corporate patronage provides a new model for a practical concern: *open source project members must make a living*. The traditional view of open source projects is that volunteers contribute work in their spare time. This volunteer model is compelling because it explains how project members can exist doing work for free—the open source project is a part-time hobby. However, we believe that a new model, that of corporate patronage, is starting to become very common on successful open source projects. Some examples include, Linus Torval's "second-in-command" Alan Cox's long term employment by Red Hat (the largest U.S. Linux distribution vendor) and Sun Microsystems employment of nearly the entire core development team of the NetBeans project (plus providing commercial quality Quality Assurance for the project). More recently, the Eclipse project (at www.eclipse.org) is supported by employees from a wide group of companies including IBM, Merant, QNX, TogetherSoft, Rational, RedHat, SuSE, and Borland [3].

Traditional project management is the art and science of balancing cost, schedule, and performance toward some focused outcome. We observed a strong bias in open source projects toward *fixing cost and performance with time as variable*. Open source projects use issue tracking tools, such as Bugzilla, to track enhancements (performance) and the programmers working on them (cost). A major release of the project's software may be linked to several enhancements being completed. Detailed schedules or PERT charts are absent from open source projects, although general roadmaps are common.

Table 3: Open source collaboration tools

Tool Category	Name (Table 1 occurrence frequency)
Web Portal	Custom(9), SourceForge.net(1), SourceCast(1)
Source Code Control	CVS(11)
Code Viewers	ViewCVS(4), LXR(4), CVSWeb(3), Bonsai(2), CodeWeb(1)
Mailing List Management/Archive	Mailman(5), Ezmlm(3), MHonArc(2), Majordomo(1)
Bug/Issue Tracking Tools	Bugzilla(4), GNATS(2), DebBugs(1), Perlbug(1), RequestTracker(1)
Test Support Tools	Tinderbox(2), DejaGnu(1)
Instant Messaging	IRC(2)

2.4 Process and the walled project server

There is an important distinction in open source projects between the policies of visibility (including both access management and intellectual property disposition of the shared software assets) and the policies of engineering practice (including particularly architecture, tool use, process, and roles of people). The fact that open source projects, in theory, allow anyone to view and copy the source code (or even start a competing open source project) raises an interesting question: What do the leaders of an open source project actually control? In a nutshell, they control the composition, configuration, and information flow in and out of their project's server.[4] And in this way they can exercise tight control over the engineering practices of the project—not by limiting the behavior of individual developers in their own personal space, but by limiting the kinds of transactions developers can make with the persistent project state on the server.

Metaphorically, the project server is surrounded by a *wall*. This wall, embodied by the web portal and open source collaboration tools, is designed to *maximize* outgoing information flow and simultaneously to *strictly limit and control* incoming information flow. The critical processes of the project are enacted behind the server wall. These processes include code commits, documentation management, configuration management, builds, regression tests, and the like. This model is shown pictorially in Figure 2.

The walled project server thus physically embodies critical aspects of software best practice (principally process, build management, design record, and architecture) as defined by the project leaders. This enables effective engineering management despite the fact that the engineers are self-selected and largely self-managed. This enables combining managed rigor in project-level practice with a complete lack of restraint on the client-side developer space.

2.5 Boot-strapping

Table 3 categorizes the collaboration tools reported in Table 1. With the notable exception of the SourceCast web portal, all the tools listed are open source. It is often observed [18, 17] that many open-source projects engage in *boot-strapping*—the use of open source tools on open source projects. Indeed, we find boot-strapping common for collaboration tools. The software development tools (not reported in Figure 1) used within these projects were also found to be predominantly open source. An interesting exception was the use of Sun's Java development tools within the Jakarta and NetBeans projects. These tools are easily and

[4]Here we use server to refer to the logical grouping of all server-based open source tools and its presence on the Internet, not any single computer.

"freely" available, but are not open source (they do not meet the Open Source Definition [36]).

The composition of the actual project web portals that encapsulate open source collaboration tools, were also predominantly open source. Apache appeared to be the web server selected for every web portal surveyed. Web server side tools such as MySQL, PostgreSQL, PHP, Perl, and Python were common (e.g, PHP using PostgreSQL make up the bulk of the Soureforge.net portal, while Perl using MySQL make up the bulk of Bugzilla).

While boot-strapping is often rationalized on the basis of ideology, in fact it has two significant effects. First, it lowers the barrier to entry for new participants in an open source project, enabling participants to create their personal (client-side) developer sites at low cost and without reliance on particular products. Second, it enables developers to fluidly shift their attention from tool use to tool development and repair. This movement up and down the tool food chain reinforces the developers-as-users principle.

A seeming exception to the principle of "tool openness" is the use of CollabNet's proprietary SourceCast web portal by the NetBeans project. This web portal does not raise an entry barrier, however, because it is built around familiar open source tools. In addition, the proprietary portion of SourceCast exists only on the project server—the tools used by project participants on their individual computers remain open source (or at least cost-free). This may be a significant distinction. It also raises the question of whether ideology is the driver or perhaps just a *post hoc* explanation for many successful open source practices. Nonetheless, it is clear that ideology within many open source projects today means that only open source tools will be adopted. Indeed, comparing project counts in Table 2, shows evidence of this ideology—Tigris.org, like NetBeans, is hosted by CollabNet's proprietary SourceCast.

2.6 Incrementality and gentle slope

Most project portals offer resources to help potential new participants quickly reach the point of becoming visible and acknowledged contributors to the project. We can hypothesize that this desire to create a "gentle slope" learning curve [this term due to Michael Dertouzos] has yielded a *de facto* consensus on infrastructural tools such as Apache, CVS, and others. It also increases the payoff for developers down-shifting in the food chain from tool user to tool developers.

In addition to tool selection, "gentle slope" is reinforced by the communication practices noted above, in which increments of contribution can be made by participants, enabling other participants to build on the results and providing an explicit and objective record of the personal role of the contributor.

To the extent that it can be achieved, the incremental "gentle slope" model has significant effects for a project and its participants. It enables certain categories of participants to gracefully engage and disengage in a project without adverse consequences to the project. The entry model for many projects reinforces this—with new participants encouraged to review, debug, and document existing code, contributing changes in the form of mailing list posts for the attention of those with commit privileges. The *de facto* tool consensus reduces the barrier for developer-side tooling and for acquiring skills to interact effectively with the project server. As the new participant begins to do more work with the source code, such as private builds, additional increments of tool investment are required. The point is that new participants do not incur significant risk with respect to their time and resources in order to engage, perhaps experimentally, in collaborating on a project.

2.7 Evolutionary focus

The projects listed in Table 1 all work within what would conventionally be regarded as software maintenance and evolution phases of the software lifecycle. Rarely, if ever, would any of these projects create a new version of the software without reuse of an existing code base. Indeed, it can be hypothesized that there is sensitivity in open source practice to architecture design. The long-standing success of gcc and Linux, for example, derives in many ways from architectural prescience in the initial designs—enabling both successful division of effort on the part of develop teams and long-term incremental growth in capability without excessive quality compromise. If these architectural attributes are not present in an open source project it is unlikely to enjoy long term success.

Mockus, Fielding, and Herbsleb in [31] noted this need for division of effort during a study of the Apache and Mozilla projects. They hypothesized that "For projects that are so large that 10-15 developers cannot write 80% of the code in a reasonable time frame, a strict code ownership policy will have to be adopted to separate the work of additional groups, creating, in effect, several related OSS projects." In addition to code ownership they find that explicit development processes and code inspections may also support division of effort. As noted above, we believe, that a good software architecture is also critical to effective division of effort and an ability to support long term software evolution.

We conclude this section by reiterating several key points about the open source development method.[5]

> *Software must be distributed under an Open Source Definition (OSD) compliant license allowing free redistribution, source code access, and derived works.*
>
> *Co-location is never assumed.*
>
> *Tool mediation is the norm.*
>
> *Public web portals act as the project focal point.*
>
> *Projects are lead/managed via meritocracy.*
>
> *Open source project members must make a living. The traditional volunteer model of open source contribution is being augmented by corporate patronage of the project.*
>
> *In general, time is variable with fixed cost and performance.*
>
> *A "walled server" physically embodies software best practice.*
>
> *Boot-strapping is common due to ideology and keeping a low barrier of entry.*
>
> *An incremental "gentle slope" learning curve exists to allow easy integration and acknowledgement of new project members.*
>
> *An open source program's software architecture is critical to effective division of effort and an ability to support long term software evolution*

3 The social context of open source

One of the most striking features of open source is the collaborative nature of the communities, which involve clusters of software developers working at different locations and organizations sharing source code over the Internet to develop and improve programs. How do these loosely coupled communities work? How do they organize themselves to generate software capable of threatening the position of the well-established leaders in the software industry? In order to shed light on the ways of organizing and getting things done in open source, it would be essential to uncover the motivational, social and political processes underlying the organization and functioning of these virtual communities.

Despite the increasing notoriety of open source software, little research has been conducted on processes underlying the organization and functioning of open source communities. Lerner and Tirole [25] explored the economics of open source and the set of incentives that motivates developers to voluntarily participate

[5]We will use quoted italic paragraphs to highlight significant findings for the remainder of this document.

in both major task and field support, highlighting the extent to which labor economics and industrial organization theory can explain these issues. McGowan [28] examined the legal implications of open source based on the literature on intellectual property rights, contract law, and theory of the firm. Furthermore, he explored key motivational and organizational aspects of the movement concluding that open source software projects are more likely to succeed when the need for coordination and the cost of social interaction through the mailing lists are relatively low. On the other hand, complex projects present a harder challenge for open source communities by imposing an increasing need for coordination, some degree of hierarchy, and behavioral norms to support it. Metiu and Kogut [30] focused on the implications that open source has for the process of development in the global software industry in terms of the three key drivers of this process: "coordination of modularized tasks, communication, and social context." Finally, Lakhani and von Hippel [24] empirically examined motives for developers to participate in providing the "mundane but necessary" task of field support revealing that "information providers are largely rewarded by benefits directly received from a related task." This latter study represents a sound attempt to explore organizational processes in open source communities through a detailed examination of mailing lists.

This section intends to advance our understanding of the social context of open source communities by examining two of the processes that appear to be essential in the functioning of these communities: *communication* and *intragroup conflict*. Indeed, the anecdotal literature on open source has suggested that both communication and conflict play fundamental roles in the functioning and development of open source communities [42, 39]. For instance, open discussions about appropriate licensing models have contributed to forge a sense of belonging to a particular community within variants of open source. Open disagreements about features of a program or pieces of the code have given rise to either more robust software development or code-forking.[6] Furthermore, the anecdotal literature has also suggested that members tend to increase their reputation and prestige in the open source community by making contributions that help improve or replace conflicting pieces of the code or features of a program [42]. All in all, we explore the effect of different types and levels of conflict on performance in open source software development as well as the structural consequences of conflict resolution on the reputational network of its communities.

3.1 Characterization of open source communities

The increasing speed and widespread diffusion of Internet access in the nineties has led to a dramatic acceleration of open source and the free software movement. The availability of neat open source code on the Internet (e.g., Linux, Apache, Perl) attracted software developers who liked to play around with technology. Some of them stayed on, engaged in the mailing list discussions about strengths and weaknesses of the code, and also began contributing back to the project. The distinctive nature of software as a product is what makes it possible for open source communities to operate almost exclusively through mailing lists and other previously discussed collaboration tools. Unlike other products, the process of software development and distribution can be shared over the Web. In a sense, software is notably similar to information considered as a product: once someone writes the source code, the cost of transmitting it is next to nothing.

The literature on software development has indicated that open and evolutionary models of development provide a context in which improvements occur dramatically faster compared to traditional proprietary and closed models. While the process of software development or "coding" remains an essentially solitary

[6]Rosenberg [39] defines forking as "the splitting of a software project into two or more branches, which continue in parallel, but incorporating different code and probably addressing different problems for a different set of users." Code-forking occurs when normal disagreements about the development process lead unsatisfied developers to leave the project and initiate their own version of the program. Unsatisfied developers will need to attract enough developers to sustain a new community and successfully run the forking project. More important, the emergence of inconsistent versions of the project will reduce the opportunities for rapid diffusion by subdividing the user's base. A code-fork is simply an extreme form of derived work (recall that derived works must be allowed by all open source software licenses).

activity, open and evolutionary models create an environment in which a potentially unlimited number of developers can provide feedback about software design, pieces of the code, bug-spotting, or features of the program. As Raymond has eloquently described in [37], “given a large enough beta-testers and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone,” or, in other words, “given enough eyeballs, all bugs are shallow.” In fact, open models of development radically contrast proprietary models in which only a few developers work clustered together in closed projects doing the tedious scrutiny of bugs and solving development problems for long periods of time before releasing new versions of the product.

While the widespread diffusion of the Internet was a necessary condition for the rise of open and evolutionary models of development, it was by no means a sufficient condition. Other factors related to the development of a set of norms aimed at attracting a large base of beta-testers and co-developers also played a fundamental role in the emergence of open source communities. Among them, “release early, release often” and “treat your users as co-developers” can be considered as fundamental norms of open source communities [37]. In fact, these norms have contributed to attract users who liked to play around with the source code and engaged in the mailing list discussions about software design and development. Frequent new releases incorporating pieces of users’ feedback further stimulated them by providing improved versions of the software and expectations of ego-satisfying rewards. These norms, along with the intrinsic nature of the open model of development (emphasizing open disclosure of information), guarantee that, despite some rare private communications among developers and face to face meetings, the vast majority of the group interactions takes place through the publicly accessible mailing lists.

During the last few years, the volume and diversity of contributions expanded sharply and new projects emerged giving rise to a bunch of open source communities united by mailing lists. Few software developers follow the whole movement, whereas most of them engage only in specific aspects of the software according to their interests or expertise. By spontaneously allocating members’ contribution to different activities such as solving bugs, improving code, adding features into programs or providing technical support, these open source communities virtually establish a sort of division of labor within the network.

While some features or programs are very popular and have a large number of software developers working regularly on their source code, others may be more ephemeral and tackled only by people who need to solve a particular problem. Communities seem to emerge and evolve over time through very social and political process that unrolls almost exclusively through e-mail messages. In particular, a former participant described the process as follows: “One person makes a suggestion. Others may agree. Someone may squabble with the idea because it seems inelegant, sloppy, or, worst of all, dangerous. After some time, a rough consensus evolves. Easy problems can be solved in days or even minutes, but complicated decisions can wait as the debate rages for years” [42]. Remarkably, communication and intragroup conflict seem to be at the center of the process of software development in open source communities, especially when they face complicated decisions and consensus does not easily evolve.

3.2 Communication: electronic flows of information

The rise of open source communities could not be possible without the significant advances in communication technology, which enables an accessible and varied array of communicative practices. Communication and coordination in open source communities is extremely dependent on the Internet’s collaborative technologies, such as e-mail lists, newsgroups, web sites.[7] After conducting a preliminary analysis of a small sample of messages submitted to a OSS development list, three different flows of information were broadly

[7]An extensive literature has contrasted advantages and disadvantages of face-to-face versus digital communication (see [34] for a recent review) as well as the ways computer-mediated communication has been used to supplement the lack of nonverbal cues (McGrath and Hollingshead [29])

identified:

- *Field support communication.* This process involves new users or less sophisticated users experiencing problems while implementing or running a program or specific features of a program. In general, it takes only a short period of time for a sophisticated user to directly provide a solution to the problem or to send the inexperienced user to the appropriate section of the FAQ (frequently asked questions) archives. Conflict very seldom emerges from this flow of communication.

- *Task-related communication.* This process involves a number of core developers (i.e., regular contributors) working on specific areas of the software such as writing code, fixing bugs, or creating new features. The main goal is to make the software more robust. However, a great deal of disagreement usually emerges around what constitutes better code or what are the more desirable features of a program. Such conflicts must be resolved in order to continue making progress in a project. Given the voluntary nature of open source communities and the need to maintain a broad base of qualified contributors, conflict resolution must carefully evolve through consensus-building and performance review. As described by a former open source participant in [42], "complicated decisions can wait as the debate rages for years." Moreover, the risk of code-forking is one of the major threats in any open source community.

- *Value congruence communication.* This communication flow generally involves key participants in open source addressing fundamental values and behavioral norms embedded in the community. Messages usually refer to fundamental values regarding the battle between closed-source development (i.e., the proprietary models of development) versus open-source, the appropriateness of most prominent licensing variants (e.g., Freeware, BSD style, Apache style, GPL). In addition, it also deals with some critical behavioral norms regarding the appropriateness of the governance structure, the approval procedures for new pieces of the code, the process of conflict resolution, and the time and opportunity for releasing new versions of the program.

3.3 Beneficial and detrimental effects of conflict

As a group performs complex and interdependent tasks, conflicts or disagreements among group members inevitably arise. This pervasive emergence of intragroup conflict is largely reflected in the fact that conflict has been a central concept in organizational studies of group processes and performance [19, 35]. Over the years, different perspectives have evolved regarding whether conflict is detrimental or beneficial to group functioning and performance. The traditional organizational behavior literature considered conflict to have detrimental effects on performance and satisfaction, and thus, directed much of its attention to a set of mechanisms to either avoid or resolve group and organizational conflict [27]. The underlying assumption behind this position is that intragroup conflict arises because members have apparent opposing or incompatible goals [26]. More recent trends in the organizational literature, however, have shown that under some particular circumstances intragroup conflict can be functional, leading to beneficial effects on performance [22]. This line of research recognizes sources of conflicts other than opposing or incompatible goals, that is, while members may largely agree about group goals (i.e., group ends), conflicts may stem from members' disagreements about their different approaches to solving problems and task coordination (i.e., group means). Identifying circumstances under which conflict can be beneficial for groups and organizations has been one of the important challenges in the organizational literature on communication and conflict.

3.4 Conflict in open source communities

The literature on group processes has predominantly focused on two types of intragroup conflict: *task conflict* and *emotional conflict*. Task conflict refers to disagreements directly related to the substance of the task that the group is performing. Emotional conflict refers to socio-emotional and interpersonal disagreements not directly related to the task. Over the years, different authors have used more or less the same definitions to categorize these two types under different combinations of labels.

3.4.1 Task conflict

In open source communities, task conflicts emerge out of the task-related communication flow. Messages making contributions around specific areas of the software such as writing code, fixing bugs, or creating new features usually generate some level of disagreement among developers. Previous research has shown that despite some negative effects of task conflict on behavioral or attitudinal outcomes,[8] there is a curvilinear relation between task conflict and group and individual performance in face-to-face groups with nonroutine tasks [22]. That is, low levels of task conflict are related to low levels of performance, high levels of conflict are related to high levels of performance, and very high level of conflicts are related to moderate levels of performance. Two aspects of this curvilinear relation deserve particular attention. First, groups performing nonroutine tasks such as software projects benefit from diverse approaches to process or product development and from critical evaluation of alternative approaches (i.e., minimizing groupthink). And second, the fact that performance diminishes beyond some "optimal level" of task conflict has been mainly attributed to the limited cognitive capacity to adequately evaluate and integrate the multiplicity of diverse approaches. Given the context of open source communities, another important issue to be considered is the extent to which results on task conflict in face-to-face groups generalize to the context of computer-supported groups. The anecdotal literature on open source has suggested a pattern of relationship between task conflict and performance similar to the one described for non-routine tasks [42]. More important, previous research on the impacts of technology in work groups has revealed no significant effects for communication medium (i.e., face-to-face versus computer mediated communication) on the level of conflict experienced by the groups [33].

> *In open source communities, there will be a curvilinear effect of task conflict on individual and group performance generating inverted-U shaped curves.*

3.4.2 Emotional conflict

In open source communities, emotional conflicts emerge out of the value congruence communication flow. Messages addressing fundamental values and behavioral norms embedded in the community may generate some level of disagreement among developers. Emotional conflicts seem to be more likely to occur in young open source communities, in which some of fundamental values and behavioral norms are not so firmly established compared to mature communities. In addition, because the development process almost entirely unrolls through mailing lists, there is little room for demographic factors to lead to increasing levels of emotional conflict.[9] These factors are not easily observable in open source communities where participants usually refer to themselves by simply using their user name or a nickname. Previous research in traditional organizational settings has shown increasing levels of emotional conflict to be associated with negative behavioral and attitudinal outcomes such as frustration, psychological withdrawal, low satisfaction,

[8]In particular, lower level of satisfaction, liking of other group members, and intent to remain in the group [22].

[9]The organizational literature on demography has found variables such as race, and tenure diversity to have a significant positive relationship with emotional conflict (e.g., [35]).

dislike of other group members, and intent to quit the group [22]. In the context of voluntary open source communities, where developers are free to leave at any time, increasing levels of emotional conflict would be directly related to members' turnover. Similar considerations as those presented for task conflict are also applicable to emotional conflict concerning the extent to which results on emotional conflict in face-to-face groups generalize to computer mediated groups.

In open source communities, increasing levels of emotional conflict will be positively associated with lower performance, increasing attempts at code-forking, and developer turnover.

3.5 Conflict resolution and actor position in the reputational network

In addition, type and level of conflict may have significant influence in shaping the social structure of the open source communities. Software developers usually engage in heated conflicts around the essential tasks of writing code, solving bugs, or creating new features; conflicts that must be eventually solved to allow for further advances in the development project. In voluntary communities, conflicts around essential tasks provide opportunities for qualified developers to display their programming skills and come out with sound contributions that would eventually enable the community to overcome critical problem. If after an exhaustive evaluation, the new contribution is proven to be a better solution to the problem, then the community will reach a rough consensus based on enhanced software performance. Moreover, the developer who suggested such a sound contribution will increase his or her reputation or prestige within the social structure of the community [42, 25]. An analogy could be established with previous research conducted by Burkhardt and Brass [15] on the effect of change in technology on organizational structure. In a study of the introduction and diffusion of a computerized information system, they found that those who adopted and mastered the new technology first increased their power and centrality to a greater degree than those who adopted later. The literature on social networks has been largely concerned with the identification of the most important and prominent actors in a social network or community. Among the various measures attempting to describe the actor location in a social network, degree of prestige provides a relatively more refined concept to refer to actors who become the object of extensive ties. Software developers with significant accomplishments in solving conflicting tasks are more likely to become more extensively cited or referred to in subsequent interactions, and therefore, to enjoy a more prestigious position within the open source community.

Individual performance in critical or conflicting episodes will be positively associated with actor prestige in the structural network of the open source community.

Finally, software developers occasionally engage in heated emotional conflicts about fundamental values or behavioral norms.[10] Given the voluntary nature of open source communities, increasing levels of emotional conflict may lead to diminishing contributions, code-forking, and turnover with potentially disruptive consequences in the structural network of the community. This is particularly true in the case of code-forking when a significant number of unsatisfied software developers decide to create a new community to run their own version of the software project. Code-forking and massive turnover in open source communities may have network consequences similar to those experienced after downsizing and massive lay-off in more traditional organizational settings [40]. When an important number of developers stops contributing to an open source community at the same time (e.g., due to code-forking), such community may be unable to sustain high levels of performance, and moreover, it may start losing popularity and interest over time.

[10]As highlighted above, emotional conflicts tend to occur more frequently in young communities.

Massive turnover or code-forking will be associated with high levels of disruption in the social network of the community.

4 Open source business models

In this section we move away from the methods and social context of open source to focus on how open source fits into a successful business strategy.

Use of intellectual property rights, i.e., copyrights, patents and trade secrets, has made it possible for traditional software companies to raise capital, take risks, focus on the long term, and create a sustainable business model. As open source companies give their software products away for free with unlimited rights for redistribution, the obvious question is: *how do open source companies make money?*

Open source business models rely upon shifting the commercial value away from the licensing of software products to generating revenue via the *product halo*. The product halo includes ancillary services like systems integration, support, tutorials, and documentation. Focus on the product halo is rooted in the belief that unlike typical manufactured goods, the value of software lies mostly in the value-added services and not in the product or any intellectual property that the product represents. A pure open source business model accepts the premise that the value of software products approaches zero in the fast-paced, highly customized, ever-changing world of information technology. As Raymond in [38] states, “Software is largely a service industry operating under the persistent but unfounded delusion that it is a manufacturing industry.” However, as we shall see, not all open source companies are pure in this sense and many don’t fully share Raymond’s point of view about the nature of the software industry.

We surveyed several open source companies and propose a set of viable open source business strategies. Our work is not the only effort in this area. Behlendorf in [17] proposed market evaluation and careful goal analysis when deciding to use open source in the business context. Feller and Fitzgerald in [18] divide open source companies into two categories “pure-play” and “hybrid”. Pure-play companies have a business model focused solely on open source, hybrid companies mix open source and proprietary business models. We find this model useful, but too simple to adequately describe current open source businesses. Raymond in [38] advocates seven open source business models but overly cute business model names like “widget frosting” and a weak grounding in concrete business examples hinders viability analysis of these models. Our observed open source business models are as follows:

- *Service Provider*. Companies using this business model sell technical support and enhancements for specific open source software. This is the most common, and to date the most successful, business model observed. What is actually being sold is not the bits, but the value added by assembling, testing, packaging, and documenting open source software. The services also include free installation support and the provision of options for continuing support contracts. This model includes selling services to companies engaged in the symbiotic model (described below).

  This model is applied by most of the major Linux distributors. For example, as an extremely successful Linux vendor, Red Hat provides a bundled open source Linux distribution and charges for services such as frequent updates, support, and training. Cygnus Solutions, an early commercial provider of open source based software development tools, support and custom engineering (including GNUPro Toolkits and embedded Cygnus operating system), which has been acquired by Red Hat, has successfully used this model since 1989.

  This model has not been successful for Mandrake (a French company), another major Linux distribution vendor, which recently started trying to solicit donations [6] to help the cash flow of the company.

- *Add Proprietary Value*. In this case, an open source project provides limited capabilities for free but a proprietary (with a license fee) version adds significant value to the free software. There are two different cases of this model. First, a company may release its existing software as open source. In March of 1999, Apple released the core layers of the Mac OS X operating system as an open source operating system called Darwin [2], and built the proprietary Mac OS GUI on top of Darwin. The second case is when a company chooses an outside open source project to support (patronage) and adds value in a proprietary version of the software. For example, IBM's recent release of $40 million worth of software to the Eclipse project [23] was done with the goal of using Eclipse software as the basis of IBM's next generation of WebSphere Studio products. Sun also provides similar support to the open source NetBeans project, while selling a enhanced proprietary version of NetBeans called Forte for Java. This business model is only possible when the open source software does not mandate copyleft in its license.

- *Marketing Gimmick (Market Positioning Strategy)*. In this model, a company tries to create or maintain a market position for proprietary software by attracting customers via an open source project. It is the common case that open source client software enables sales of server software. The Enhydra open source project (reported in [21]) used this model until it was finally terminated as an open source project. This model does not appear sustainable in the long term.

- *Symbiotic Model*. Using this model, a company supports an open source project with programmers or equipment (e.g., bandwidth or servers) with an aim of establishing a symbiotic relationship with some other aspect of the business (usually selling hardware or software). OEone [8] has created an entire operating system around the Mozilla project and has contributed vast amounts of software back to the project. One of the motivations of IBM's Linux mainframe port [10] was to sell mainframe hardware to customers interested in Linux technology.

- *Facility Provider*. A few companies have started businesses with the purpose of supporting web portals for open source development. CollabNet and OSDN work within this business model, but with slightly different approaches. CollabNet [1] provides for fee portal hosting for both commercial and distributed commercial development (using open source software techniques). OSDN provides the SourceForge.net web portal for free open source development, but also sells a proprietary enhanced version of the SourceForge web portal [12] that companies may use for open source or commercial development.

5 When should a business develop software using open source methods?

When should a business use open source software methods over other software development methods? This section focuses on this question and proposes some possible answers. Like any major business decision the selection of a software development method has many facets and we have, in all likelihood, overlooked several aspects of this decision that would be critical to many individual companies.

We begin by examining advocated claims in this area. Raymond in [38] posits the following discriminators for the selection of open source over closed source:

1. Reliability / Stability / Scalability are critical

2. Correctness of design and implementation cannot be readily verified by means other than independent peer review.

3. The software is critical to the user's control of his / her business

4. The software establishes a common computing and communications infrastructure
5. Key methods (or functional equivalents of them) are part of common engineering knowledge.

Raymond further notes that "open source seems to make the *least* sense for companies that have unique possession of a value generating software technology which is relatively insensitive to failure, which can readily be verified by means other than independent peer review, which is not business-critical, and which would not have its value substantially increased by network effects or ubiquity."

Item 3 in the above list is tied by Raymond to overall strategic business risk: "the brutal truth is this: when your key business processes are executed by opaque blocks of bits that you can't even see the inside of (let alone modify) *you have lost control of your business.*" Raymond further asserts that selecting single-vendor closed source over open source for mission critical business software will soon be viewed as fiduciary irresponsibility.

Raymond believes that applications and some middleware will remain largely closed source while nearly all infrastructure software will become open source. The interesting aspect of his analysis is that over time definitions of what fits into these categories changes, hence most software begins in the application category and slowly (or quickly) migrates into the infrastructure category. Therefore, "In a future that includes competition from open source, we can expect that the eventual destiny of any software technology will be to either die or become part of the open infrastructure itself. While this is hardly happy news for entrepreneurs who would like to collect rent on closed software forever, it does suggest that the software industry as a whole will *remain* entrepreneurial, with new niches constantly opening up at the upper (application) end and a limited lifespan for closed-IP monopolies as their product categories fall into infrastructure."

We, in the whole, agree with Raymond's discriminators,[11] but offer the following other considerations:

- *Corporate Culture Clash*. Is the culture of your company able to cope with giving over control of ongoing software development to a meritocracy that can only be influenced indirectly by corporate management? This may be a serious issue to corporate leadership.
- *Customer/Stockholder Culture*. Are your current customer base and your stockholders ready to deal with your software as open source? If your customer or stockholder base is likely to be alienated this could cause your market or management support to dry up.
- *Boot-Strapping*. Are you willing to give up commercial development tools (including your investment in staff training) to use open source development tools? If your company has invested tremendous amounts of money in commercial software development tools and training this could be an issue.
- *System Administration Talent*. Does your company have the talent to support and operate an open source development project? This focuses on the complexity of open source collaboration and development tools. These tools are time-consuming to setup and often require source code changes to tailor them fore use on a specific project.
- *Code-Fork Threat—Losing Control*. What will your company do if another group code-forks your software and begins a competing project? What steps do you plan to take to keep the developer community at your version of the project (e.g., pay salaries for the vast majority of the key developers)? What are you going to do to ensure the community never sees the need for a code fork? The serious threat of losing control of "your" software may be a serious issue to many companies.

[11]We suspect, however, that in practice our decisions about when open source makes business sense may be more conservative than what Raymond would advocate.

6 Implications for future research

In this section we propose four areas for future research: open source patronage, competition between open source and proprietary software, open source use as Off-The-Shelf (OTS) software, and transplanting the open source software development method. In general, we advise a strong focus on empirical investigation of actual open source software projects and companies. This is due to the large amount of ideology present in much of the open source literature—separating reality from hype can be difficult.

Observe and measure open source first hand; don't just read about it.

6.1 Open source patronage

Although discussed several times in this paper, patronage of open source projects by companies or government is not well understood. Should companies or governments support open source projects? To what extent? We suggest that case studies of open source projects with strong corporate or government patronage be performed. Some possible case studies include:

- NetBeans (supported by Sun Microsystems)
- Eclipse (supported by IBM, Rational, and others)
- Linux (supported by Linux distribution companies)
- Linux (selected by China and considered by Germany as a government standard)

We believe understanding the true impact of open source patronage to be critical to the overall future of the worldwide software industry.

6.2 Competition between open source and proprietary software

This area primarily affects software producing companies (e.g., Microsoft). What should a software producing company do if a viable open source product begins to compete with its products? This is a difficult question and experience to date provides no clear answers. Historical examples illustrate two extremes. First, the GNU Compiler Collection (GCC) has provided a useful programming language compiler for over a decade, but a thriving commercial compiler industry co-exists with the open source GCC product. At the other extreme, the Linux operating system is in the slow process of decimating the commercial Unix market. Today, most commercial Unix operating systems (e.g., Solaris, HPUX, SCO Unix), if they still exist, are provided free of charge. Competition from Linux has drastically altered the commercial Unix market of the early 1990's, it is migrating toward a commodity service-based market. We suggest an empirical study of several commercial software markets that have viable open source products competing with the commercial offerings. Some software markets of specific interest are:

- Programming language compilers (open source product: GCC)
- Operating systems (open source products: Linux, Free/Net/Open/Darwin-BSD)
- Web servers (open source product: Apache)
- Integrated development environments (open source products: NetBeans, Eclipse)

In addition, investigation into various industry attacks on open source software (e.g., Microsoft's campaign against open source—specifically the GPL license [7]) may uncover the aspects of open source most threating to the established software industry.

6.3 Open source as off-the-shelf software

By far the most common use of open source software by industry today is as off-the-shelf (OTS) software: Open Source OTS (OSOTS). Use of Commercial OTS (COTS) software has been common in industry since the late-1960s, but recently OSOTS has become widespread as well. When should a company select OSOTS over COTS? What are the cost trade-offs involved with such a selection? Research in to these questions is underway at the Software Engineering Institute (i.e., [21]) but further work is needed. In this area we, in general, propose that methods and techniques for the application of COTS software need to be broadened to include and consider OSOTS.

6.4 Transplanting the open source software development method

Can open source development methods be successfully applied to proprietary software development? A critical step to answering this questions is understanding what is the impact of adding traditional management oversight into the open source method. Can the open source method, proven successful when led by meritocracy, work when controlled by traditional business management? We recommend a focused case study of CollabNet (www.collab.net), a company providing for fee open source-like web portal hosting to commercial companies. CollabNet has worked with Oracle, HP, Compaq and other companies [1] to use open source-like software development methods for the development of proprietary commercial software.

7 Conclusion

Today, no business producing or dependent upon software can ignore open source. We *strongly* believe that open source is not the end of the commercial software industry but it is clearly changing some areas of the industry's landscape. We have provided an overview of the open source software development method, investigated the social context of open source, proposed five possible open source business models, and evaluated when a business should choose to use open source methods over other approaches. Finally, we proposed several areas of possible future research with a strong recommendation to focus on empirical investigation and measurement of actual open source software projects and companies due ideological bias of much of the open source literature.

References

[1] CollabNet Customers. `http://www.collab.net/customers/`. Current May 2002.

[2] Darwin Project. `http://developer.apple.com/darwin/`. Current May 2002.

[3] Eclipse Consortium. `http://www.eclipse.org/org/index.html`. Current May 2002.

[4] Free Software Definition. `http://www.gnu.org/philosophy/free-sw.html`. Current Apr. 2002.

[5] GNU General Public License. `http://www.gnu.org/licenses/gpl.html`. Current Apr. 2002.

[6] Mandrake Voluntary Contribution Page. `http://linux-mandrake.com/donations/`. Current May 2002.

[7] Microsoft executive says linux threatens innovation. `http://news.cnet.com/investor/news/newsitem/0-9900-1028-4825719-RHAT.ht%ml?tag=ltnc`. Current Apr. 2002.

[8] OEOne.

[9] Open Source Initiative website. `http://www.opensource.org/`. Current Apr. 2002.

[10] Operating Systems for IBM Mainframe Servers.

[11] Reporting security problems with Apache. `http://httpd.apache.org/security_report.html`. Current May 2002.

[12] Source Forge Enterprise Edition. `http://www.vasoftware.com/products/sourceforge_enterprise_edition.php`. Current May 2002.

[13] The Borg meet Bill Gates. `http://www.gksoft.com/a/fun/borg-gates.html`. Current Apr. 2002.

[14] Halloween Documents. `http://www.opensource.org/halloween/halloween1.html`, 1998. Current May 2002.

[15] BURKHARDT, M. E., AND BRASS, D. J. Changing patterns or patterns of change: The effects of a change in technology on social network structure and power. *Administrative Science Quarterly*, 35 (1990), 104–127.

[16] CUSUMANO, M. A., AND YOFFIE, D. B. *Competing on Internet Time: Lessons from Netscape and its battle with Microsoft*, first ed. Touchstone, 1998.

[17] DIBONA, C., OCKMAN, S., AND STONE, M., Eds. *Open Sources: Voices of the Open Source Revolution*. O'Reilly & Associates, Inc, Jan. 1999.

[18] FELLER, J., AND FITZGERALD, B. *Understanding Open Source Software Development*, first ed. Person Education limited, 2002.

[19] GLADSTEIN, D. L. A model of task group effectiveness. *Administrative Science Quarterly*, 29 (1984), 499–517.

[20] HALLORAN, T. J., AND SCHERLIS, W. L. High quality and open source software practices. To appear in *Proceedings of the 2nd Workshop on Open Source Software Engineering* (an official event of the 24th International Conference on Software Engineering), 2002. available at `http://opensource.ucc.ie/icse2002/`.

[21] HISSAM, S. A., WEINSTOCK, C. B., PLAKOSH, D., AND ASUNDI, J. Perspectives on open source software. Tech. Rep. CMU/SEI-2001-TR-019, Carnegie Mellon Software Engineering Institute, Nov. 2001.

[22] JEHN, K. A. A multimethod examination of benefits and detriments of intragroup conflict. *Administrative Science Quarterly*, 40 (1995), 256–282.

[23] JUNNARKAR, S. Big Blue's $40m giveaway to open source. `http://zdnet.com.com/2100-1104-275388.html`. Current May 2002.

[24] LAKHANI, K., AND VON HIPPEL, E. How open source software works: Free user-to-user assistance (working paper 4117). *MIT Sloan School of Management* (2000).

[25] LERNER, J., AND TIROLE, J. The simple economics of open source (working paper). *Harvard University, Harvard Business School* (2000).

[26] LEVINE, J. M., AND MORELAND, R. L. Progress in small group research. *Annual Review of Psychology*, 41 (1990), 585–634.

[27] MARCH, J. G., AND SIMON, H. *Organizations*. Wiley, 1958.

[28] MCGOWAN, D. Legal implications of open-source software (working paper). *University of Minnesota, School of Law* (2000).

[29] MCGRATH, J. E., AND HOLLINGSHEAD, A. B. *Groups interacting with technology*. Sage Publications, Inc, 1994.

[30] METIU, A., AND KOGUT, B. Distributed knowledge and the global organization of software development (working paper). *University of Pennsylvania, The Wharton School* (2001).

[31] MOCKUS, A., FIELDING, R. T., AND HERBSLEB, J. Two case studies of open source software development: Apache and Mozilla. To appear in *ACM Transactions on Software Engineering and Methodology*.

[32] NETCRAFT. Market share for top servers across all domains August 1995 - April 2002. `http://www.netcraft.com/survey`. Current May 2002.

[33] O'CONNOR, K. M., GRUENFELD, D. H., AND MCGRATH, J. E. The experience and effects of conflict in continuing work groups. *Small Group Research*, 24 (1993), 362–382.

[34] O'MAHONEY, S., AND BARLEY, S. R. Do digital telecommunications affect work and organization? the state of our knowledge. *Research in Organization Behavior*, 21 (1999), 125–161.

[35] PELLED, L. H., EISENHARDT, K. M., AND XIN, K. R. Exploring the black box: An analysis of work group diversity, conflict, and performance. *Administrative Science Quarterly*, 44 (1999), 1–28.

[36] PERENS, B. Open Source Definition. `http://www.osdn.com/osdocs/01/01/15/1818201.shtml`, 1997. Current Apr. 2002.

[37] Raymond, E. S. The cathedral and the bazaar. `http://www.tuxedo.org/~esr/writings/cathedral-bazaar/`, May 1997. Current Apr. 2002.

[38] Raymond, E. S. The magic cauldron. `http://www.tuxedo.org/~esr/writings/cathedral-bazaar/`, Aug. 2000. Current Apr. 2002.

[39] Rosenberg, D. K. *Open Source: The unauthorized white papers*. M&T Books, 2000.

[40] Shah, P. P. Network destruction: The structural implications of downsizing. *Academy of Management Journal*, 43 (2000), 101–112.

[41] Torvalds, L., and Dimond, D. *Just for Fun: The Story of An Accidental Revolutionary*, first ed. HarperCollins Publishers, Inc, 2001.

[42] Wayner, P. *Free for all: How Linux and the Free Software Movement undercut the high-tech titans*. HarperCollins Publishers, Inc, 2000.