San José State University
SJSU ScholarWorks

Faculty Publications | Organization and Management, Department of

2005

Contrasting Community Building in Sponsored and Community Founded Open Source Projects

Joel West
San Jose State University, joel.west@sjsu.edu

Siobhán O'Mahony
Harvard Business School, somahony@hbs.edu

Follow this and additional works at: http://scholarworks.sjsu.edu/org_mgmt_pub

Recommended Citation
Joel West and Siobhán O'Mahony. "Contrasting Community Building in Sponsored and Community Founded Open Source Projects" *System Sciences* (2005): 196c. doi:10.1109/HICSS.2005.166

This Article is brought to you for free and open access by the Organization and Management, Department of at SJSU ScholarWorks. It has been accepted for inclusion in Faculty Publications by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

Contrasting Community Building in Sponsored and Community Founded Open Source Projects

Joel West
San José State University
<Joel.West@sjsu.edu>

Siobhán O'Mahony
Harvard Business School
<somahony@hbs.edu>

Abstract

Prior characterizations of open source projects have been based on the model of a community-founded project. More recently, a second model has emerged, where organizations spinout internally developed code to a public forum. Based on field work on open source projects, we compare the lifecycle differences between these two models. We identify problems unique to spinout projects, particularly in attracting and building an external community. We illustrate these issues with a feasibility analysis of a proposed open source project based on VistA, the primary healthcare information system of the U.S. Department of Veterans Affairs. This example illuminates the complexities of building a community after a code base has been developed and suggests that open source software can be used to transfer technology to the private sector.

1. Introduction

As the term is commonly used today, "open source" encompasses both an intellectual property strategy and a development methodology. However, this term masks key differences in the ways in which open source projects are created. Most of the focus of open source research has been on open source projects such as Linux, Apache or GNOME which are founded by a "grass roots" community of user-developers.

However, more recently, companies and even governments have sought to obtain the benefits of open source by releasing their proprietary source code to create new open source projects. Examples of this include IBM (Eclipse), Sun (OpenOffice) and Netscape (Mozilla). Conversion of established proprietary projects face different challenges than previous research on community-founded projects.

Thus, we examine the different tensions that community initiated and sponsored projects face when trying to build an open source community. We consider the challenges in starting and managing such projects, focusing on the necessary incentives and organizing mechanisms. We also consider the intellectual property implications associated with these different approaches.

From this analysis, we consider potential sponsorship by the U.S. Federal government, which is a major creator, sponsor, and consumer of information technologies. The government has many systems that could be converted into open source projects, either to support technology transfer goals or to incorporate external contributions to improve system quality. We analyze one such candidate, the VistA health care information system from the United States Department of Veterans Affairs (VA) to illustrate the key issues sponsors face in creating spinout open source projects.

2. Models of Community Development

The most familiar model of open source development, popularized by the Linux kernel project, is publicly initiated by one or more individuals who recruit developers to contribute to software that is still in its infancy. In this model, employment relations and reporting relationships do not guide the development of code.

Since Netscape publicly released the source code for its Internet browser, Mozilla, in 1998 an increasing number of public and private sponsors have released code created under proprietary conditions in the hope of growing a community to improve and maintain the future code base. Sponsors may intend for the community to either supplant or replace prior proprietary development efforts. After examining community initiated projects at their startup and mature stages, we turn to examine what sponsored projects do and do not share with them.

2.1 Community-Initiated Projects

Community founded open source projects are initiated by one or more individuals independent of their employment context. Familiar examples included Linus Torvalds' starting the Linux operating system, Miguel D'Icaza's initiating the GNOME desktop environment project, and the eight developers who adapted the NCSA web server to become Apache.

Community founded projects remain community managed, and are usually structured to prevent a takeover by a firm or other organization. We define a community managed project as an open source software project initiated and managed by a distributed group of individuals who do not share a common employer (O'Mahony, forthcoming). A firm may sponsor contributors, but firms typically cannot become members. Sponsored contributors must earn and sustain their role on the project under the same terms as volunteers (O'Mahony, forthcoming).

2.1.1 Startup Phase

The maturity of a project's code base in the startup phase is a critical dimension which distinguishes community and sponsored projects. Community managed projects grow organically: the community and code scale in parallel. The key design decisions, prioritization and implementation approaches come from the founders and slowly scale as other developers working in tangential areas join the community.

For example, project members may 'test' a newcomer's ability to contribute to a project by evaluating their ability to articulate a contribution to the code base. Potential members create 'joining scripts' specifying a unique contribution to become an accepted contributing member of the project (von Krogh, et al, 2003). In this manner, the complexity of the code grows as new developers join.

As a result, new community members are incorporated early in the design phase. This can create confusion and ambiguity in the short term, but enable a more robust project architecture and greater individual commitment in the long term. For example, very early in the development of Linux, Torvalds relied on others to design and implement crucial networking libraries, developing the "lieutenant" system of senior technical leaders that remains today (Moody, 2001). Early participation in the design phase can increase individual commitment to the project.

A modular project architecture may also help a project scale in a decentralized fashion. Baldwin and Clark (2003) show that projects that are more modular are better positioned to recruit new contributors. A greater number of modules can offer more opportunities for recognition, which can further enhance contributors' motivations (Baldwin and Clark, 2003).

Contributors to community initiated projects are motivated by a complex set of personal and career concerns. Lerner and Tirole (2002) argue that contributors to community projects participate to improve the visibility of their skills in the open labor market. A more recent survey of SourceForge developers shows that most contributors to community projects are interested in building their skills and solving technical problems and that career benefits are of secondary concern (Lakhani and Wolf, 2003).

The choice of license, intent and control structure can also affect the ability of a community project to attract developers. Linux, Apache, GNOME and the Debian Linux distribution crafted clear statements about the project's intent and its boundary with the commercial world. These projects used the Berkeley Software Distribution license (BSD) or the GNU General Public License (GPL), which have been in use for a long time: their commercial limitations and affordances were well known. This helped assure contributors who might otherwise worry that their work might be distributed under proprietary conditions (Moglen, 1999; Stallman, 1999) or go unrecognized.

And finally, the reputation of a project's founder is also critical to recruiting developers[1]. Individuals who were highly respected in the programming community founded the community projects listed earlier. A solid track record can enhance potential contributors' confidence in the project's viability.

The main disadvantage of the community founded approach is that startup costs — both organizational and technical — are fully born by the community. In its purest form, volunteers establish structure, collaboration technologies, governance and define roles for participation all while trying to build a working 1.0 system. Not surprisingly, only a few community open source projects ever build a complete system or establish a thriving online community. Two recent surveys of the SourceForge.net website, the largest repository of open source software, demonstrate that community project success is fairly skewed (Krishnamurthy, 2002; Healy & Schussman, 2003). Of the 46,356 projects hosted on SourceForge only 75% of projects had any software in their code repository and 95% of the projects had 5 or fewer registered contributors (Healy & Schussman, 2003). Less than 2% of projects were classified as mature.

One interpretation is that the Sourceforge hosting site serves as an incubator for fledgling ideas that might become projects. In such a forum, starts and failures are, naturally, more visible. Not all of these projects are of the same scope nor will they all share the same potential to scale as projects like Linux. However, with an increase in the number of projects in incubation, there is likely greater competition for donated labor. Furthermore, the continued reference to the same handful of successful projects (notably Linux and Apache) suggests that there is tremendous difficulty building projects this way.

2.1.2 Mature Phase

Mature community managed projects have developed a series of major releases. They have defined membership criteria or boundaries: contributors know whether they are in or out of the project. Mature projects have adopted governance mechanisms that enable representation in commercial and legal settings. They also have an ecology of institutions that support and/or extend their work. These institutions may be non-profit organizations such as the Open Source Development Lab, firms developing complementary products, or other community projects with which they collaborate.

Mature community projects founded prior to the popularization of open source software in 1998 had an

[1] We thank Patrick Wagstrom for pointing this out.

advantage that community projects founded later lack. These projects grew out of the limelight of commercial attention. Slow growth allowed them to develop a core of contributors to diffuse project norms and help adapt to later internal and external pressures associated with growing commercial interest in open source software.

For example, when mature projects such as Apache, GNOME, and Debian achieved global recognition for their work, the number of potential contributors soon outpaced the projects' ability to welcome them in the informal capacity that marked their organic growth. Attention from investors, analysts, the media, and potential commercial collaborators brought new types of list subscribers, inquiries and contributors. Project members found some of their informal decision mechanisms inadequate for making formal commitments, guiding the project's direction, and legally representing the project (O'Mahony, 2002).

Apache, Debian, and GNOME responded to these challenges by designing formal membership criteria and application processes to manage an onslaught of new recruits. Under a deluge of applicants, Debian temporarily closed its doors to new members to design a membership process (O'Mahony and Ferraro, forthcoming). Projects also adopted more formalized governance mechanisms embodied in non-profit foundations that help oversee project direction but not day-to-day technical decision making (O'Mahony, 2002). Since 1997, 20 community managed projects have created nonprofit foundations.

Mature community managed projects evolved through a process of slow organic growth and then experienced a spike of rapid growth. What may be unique to community managed projects, is that the formal governance mechanisms designed, at their roots, reflect longstanding project norms. The governance mechanisms that emerged on projects like Apache, Debian, and GNOME were conceived to correct current problems — or to reflect the way the project was currently working and help it continue on that same path. Governance was not imposed, but emerged out of shared perceived need.

The collective design of these governance mechanisms also enabled later project entrants who did not have a voice in the project's technical design, to participate in the design of the project's organization and governance. This could have a similar effect of amplifying individual commitment to the project.

2.2 Spinout Projects

An alternative open source project model is a sponsored or spinout model (West and Gallagher, 2004). A sponsor of an internally developed software project releases its code to the public under an open source software license, inviting the external community to join the project. The sponsor could be a firm, government agency or a non-profit organization.

Two of the earliest sponsored projects were founded in 1998 when Netscape formed the Mozilla project and when IBM released its Java compiler to create the Jikes project. Table 1 lists some spinout projects.

2.2.1 Startup Phase

Why do firms release their code to form an open source project? West (2003) identifies two reasons: to win adoption, or to gain development assistance on non-critical areas. What would motivate external users to adopt the technology or make contributions?

First, a sponsored open source project starts with considerable investment from sponsors prior to the launch of an open source project, and presumably, as it progresses. This can provide a solid technical foundation for large-scale innovation. These resources reduce the risk that community members will someday find the project defunct before its software is usable.

Second, the ongoing role of a formal sponsor can reduce ambiguity and provide structure to keep the project going forward. There is clearly someone "in charge," which may enable potential contributors to more easily find their role. On-going sponsorship provides resources, legitimacy and technical capabilities to improve the odds of project success.

At the same time, introducing a community in an established project raises new challenges: technical, relational and legal. Instead of evolving with the code, the external community must be introduced *post hoc* to the existing technology. The community has not had a chance to grow with the code as it matures, but instead the infant community is presented with a large complex system that may be hard to decipher at macro and micro levels. The code will be even more difficult to learn if it was developed by a single author or a small team, in which the overall architecture and design goals remain unspoken tacit knowledge.

Even without these problems, new communities face considerable problems of motivation and coordination. The external community may have trouble developing a sense of ownership for the technology, and does not benefit from the (often crucial) intrinsic motivation that comes from building a complete system from the ground up. Also, such projects face an ongoing struggle for control of the project's future, as the sponsor and community may have different visions, goals and priorities.

In some cases, the result resembles proprietary development conducted within a glass house, in which outsiders observe but only participate at the margins of the sponsor's development efforts. This is most common when the sponsor fails to relinquish control of the project's direction and priorities. An example of this is the Darwin project, where Apple released part of

its OS X as open source but kept key components proprietary (West, 2003).

Sponsors that seek greater external participation must expend considerable time and money on community development. This includes marketing to and recruiting potential contributors, integrating their efforts into the project, and developing a governance structure compatible with commercial and non-commercial constituents. Examples of such communities that are currently under development include Sun's sponsorship of OpenOffice productivity software, and IBM's sponsorship of Eclipse development tools.

Finally, contributors to sponsored projects may be more likely to worry about their future legal rights to use, modify and distribute the code they have helped develop. Since the open source definition was drafted in 1998, the Open Source Initiative (OSI) has approved 54 licenses that meet the 10 requirements of an open source license (OSI, 2004). Fifteen of these open source licenses could be identified as firm specific and 14 as product specific. The average contributor is unlikely to understand all the nuances of these licenses regarding attribution, modification and distribution.

Even the most popular license[2], the GNU's General Public License (GPL), has not been fully tested in a court of law, leaving senior legal counsel of large I.T. companies uncertain as to its enforceability. Neither community nor corporate contributors know how their respective code contributions could be distributed should conflict ensue. Under these conditions, individual programmers are likely to rely on trust, as demonstrated by a corporate sponsor's actions to create usage and contribution rights for the community.

2.2.2 Mature Phase

Without much empirical work on mature sponsored projects to draw from, we hypothesize four scenarios.

If the sponsor retains a heavy hand controlling software development, the project may remain a closed development effort, albeit with greater transparency to outsiders such as those that supply complements. This is particularly true if (to use the terminology of West 2003), the sponsor is only "opening parts," disclosing a subset of the technologies that most users would want to build a complete working system. A related but distinct example of ongoing proprietary control can be found in "dual license" open source projects such as MySQL. The company (MySQL AB) gains revenues from selling commercial licenses to firms but also licenses their software under the GPL on terms that are acceptable to individuals and non-profit users (Välimäki, 2003). The company retains full control of its development but may be more successful in attracting contributors from those users (such as universities) who receive full value from the restricted license. Similar provisions apply to "partly open" models such as Microsoft's Shared Source (West, 2003).

Alternatively, a tightly controlled but transparent development process could help sponsored projects transition to a more open development environment. For example, Chandler, a personal information manager open source project led by the non-profit Open Source Application Foundation, is developed entirely in-house, but uses a transparent development environment specifically to betterprepare a community of potential future contributors.

A third example is when the sponsoring organization withdraws active participation and turns over control to the community. The project — whether mature or fledgling — must be supported by the community or it will die. Examples of this include both the Mozilla browser and Jikes compiler (West & Gallagher, 2004). Without a sponsor, these projects may in time become indistinguishable from community-founded projects, but today there are too few examples from which to generalize.

The fourth possibility is that an active sponsor and vibrant external community share ongoing control and responsibility for developing a complex evolving system. A successful sponsored community project would utilize a broad base of contributors, be vendor neutral and self-sustaining. Such goals are often stated by sponsors spinning out source projects, but are less often realized. Why? Perhaps sponsors are still learning how to adapt open source to meet hybrid proprietary-community needs and have yet to identify best practices in community building. Alternately, inherently conflicting goals of the internal and external actors may make such hybrids an unstable and fleeting form.

2.3 Research Questions

Our review of research on community initiated and sponsored open source projects shows that researchers know much more about community initiated projects than about sponsored projects. We have highlighted some differences between the two models that affect their ability to recruit developers, coordinate, and develop software. In doing so, we have identified an implicit trade-off between the degree of control a sponsor is ability to exert and a sponsor's ability to attract volunteer contributors.

We theorize that a more closed development process, sponsor specific legal terms, and ambiguous community leadership will impair a sponsor's ability to grow an open source community. Alternatively, firms that operate with a completely open development process, laissez-fair governance or abdicate community

[2] In mid-2004, the GNU GPL was used by 96% of 56,839 projects reporting a license on www.SourceForge.net.

leadership may unintentionally waste resources, hurt their ability to appropriate returns or advance their competition Therefore, we are interested in how sponsored open source projects can best utilize their resources while still managing to grow a diverse external community of contributors. Within this broad question, we consider how the role of individual contributors and sponsors changes over time. Our aim is to conduct comparative research on sponsored projects at different stages of development.

Next, we explore possible answers to this question by examining VistA, a federally sponsored healthcare information system that is currently a candidate for an open source project. Although VistA differs from the sponsored open source projects initiated to date, by exploring an extreme case, we can anticipate possible issues that may affect future field research.

3. VistA: Public Domain to Open Source

While Federal sponsorship of the Internet is well-known, government agencies have also paid to develop internal systems which could potentially become spinout open source projects. Such projects could improve the economic value created by tax dollars, both by transferring public technology to the private sector, and by incorporating external contributions to improve internal system quality.

Since 1985, the Federal government has deployed multiple versions of the United States Veterans Administration VistA health care information system. As public domain software, it has also been used outside the federal government, but without the multilateral collaborative development processes that characterize open source. Today, VistA is code without a cohesive development community.

In mid-2003, the authors were commissioned to study how the FOIA code could be used to create an open source project. From our prior research on open source projects, we developed a draft action plan (West & O'Mahony, 2003), which was circulated to key stakeholders inside and outside the Federal government. In response, the VA developed a plan for cooperating with an open source community. In March 2004, WorldVista (a user organization) began to discuss what such a community would look like.

3.1 History

In 1982, the Veterans Health Administration (VHA) of the U.S. Veterans Administration began an effort to automate more than 150 medical centers. It deployed its first system in 1985. Between 1985 and 1996, the VHA created an improved version with a graphical user interface, called VistA (Veterans Health Information Systems and Technology Architecture) which debuted in 1996. Today, the integrated VistA system includes more than 2 million lines of code and about 140 modules. It is used by the VHA to deliver health care to nearly 5 million patients across 1,300 sites in the U.S. (VA, 2003; Marshall 2003; Medsphere, 2003).

Because the Federal government cannot, by law, hold a copyright on software developed with tax dollars, the VistA source code is in the public domain. Since the early 1980s, nonprofit, commercial and foreign entities have used the 1966 Freedom of Information Act (FOIA) to obtain copies of the VistA source code. Through such FOIA requests, versions of VistA are in active use in Finland, Germany, Egypt and Latin America, as well as in a number of state and local health care systems in the United States.

The VistA code has been attractive to external users for two reasons. The first is the price, which is free. But the second is the system architecture: VistA provides a single integrated system that manages all patient transactions in a clinical and hospital setting, while commercial packages tend to focus on specific tasks and require manual data transfer (or duplicate data entry) as a patient moves within a healthcare system.

The VHA continues to actively develop VistA to meet its own needs. VistA has also influenced federal efforts to develop standards for a planned National Healthcare Information Infrastructure. In March 2003, various federal agencies announced a Consolidated Health Informatics initiative to promulgate VistA's record formats for use in Medicare, Medicaid and Department of Defense patient records.

3.2 Limits to the Public Domain Model

The periodic FOIA releases over the past 20 years provided static snapshots later modified by recipients to meet local needs. A few recipients applied subsequent patches distributed by the VHA, but only a handful of changes by the external developers have been incorporated by the VHA. Although VistA is in the public domain, the VHA's development processes remain closed. External developers also rarely integrate and share their code improvements among themselves.

Despite the widespread diffusion of the code, there is, as of yet, no cohesive open source community around VistA. Instead, there are dozens (if not hundreds) of disparate code variants, an extreme example of "forking". As a consequence, there is considerable duplicated effort for external users that do not benefit from the improvements of others.

While VHA programmers improve the core product, the VHA could also benefit from changes made by external users. For example, the operational VHA systems use a proprietary HP (née DEC) platform, while many external sites have modified the VistA code to use with Linux on commodity Intel-based hardware. Similarly, external efforts to abstract the VistA design could provide a single code base for the VHA and other

Federal agencies using derivative code, such as the Departments of Defense and Indian Health Services.

3.3 Creating a Spinout Open Source Project

The VHA could benefit from an open source project, but has restrictions on how it interacts with firms. The VHA cannot compete with the private sector and must avoid favoritism in its dealings. An independent non-profit foundation for VistA users could provide a mechanism for the VHA to transfer technology to governmental and commercial users, while offering a neutral forum for coordinating disparate interests.

From a technical standpoint, such a project could eliminate redundant development labor by integrating external programming efforts into a single unified copy of the FOIA source code, allowing both parties to work from a common code repository. External users would have access to the latest internal innovations and the VHA could incorporate innovations and improvements which their programmers either did not consider or did not have the resources to pursue.

At the same time, there are several key issues that must be addressed to create an effective VistA open source community. These fall into four areas: creating a technical infrastructure; establishing governance mechanisms that meet the concerns of a wide range of stakeholders; considering the role of domain knowledge; and intellectual property licensing.

3.4 Enabling Technical Collaboration

To enable collaborative software development among the sponsor's programmers and the external community, three major types of technical infrastructure are required. The first and most obvious is the *source code repository,* which allows all to view the code and trusted participants to change it. Secondly, the project needs an *issue tracking database* for monitoring software defects, "to do" lists, and desired enhancements. Finally, projects need *discussion groups,* typically implemented using e-mail lists.

Levels of authority, access rights, and participation should be scoped to increase as people gain experience with the project. For example, the Apache project distinguishes betweens user, developers, committers, lead committers and foundation members (Apache 2003). Any project must have criteria for membership, as well as opportunities for participants to assume greater responsibilities (von Krogh et al, 2003).

Finally, the project must establish formal release processes. Community projects often take a long time to herd volunteer contributors to create software that can be deployed in a usable form. Contributors from inside and outside the VA are likely to weigh the trade-offs of features, timeliness and quality differently. This tension, inherent in all spinout projects, may be attenuated on a project with a public sponsor as it faces less pressure to meet revenue-driven deadlines.

A large code base is difficult to govern as an indivisible whole, so VistA would need to be partitioned into subprojects with separate programmer communities. This would also enable decentralized and well defined access rights to control the technical architecture for different sections of the project. Such an architecture would be well suited for a federated governance system that allows all modules to be represented.

3.5 Demands of Disparate Stakeholders

A firm has a well-defined set of stakeholders, with its primary responsibility to the company's owners. The stakeholders for a non-profit organization are more complex, particularly for an open source project, which relies on interdependent individuals and organizations to create, maintain and evolve a complex technological architecture. For the VistA Project, the stakeholders would be an unusually broad group, including:

The Department of Veterans Affairs, which could gain through testing, interoperability, and improving its ability to attract and retain skilled technical workers.

Existing and potential VistA user sites. Health care organizations such as hospitals and provider networks that implement VistA will be concerned with reliability, affordability, security, and ongoing maintenance and support.

Organizations Implementing and Supporting VistA (both commercial and non-profit) provide service and support to users that are unable to access the code themselves.

VistA developers who develop and maintain the VistA code, including former VA employees now working as independent consultants.

The project's governance must bridge the long-term goals of the disparate stakeholders to the day-to-day development efforts of contributors. Directors elected to represent the interests of the disparate stakeholders could govern a non-profit foundation that holds the assets of the project.

However, directors cannot "order" volunteer contributors without losing their motivation and participation. Thus, to avoid exit and retain loyalty of technical contributors, the governance model must give voice to the expertise and views of these contributors. For example, the directors of the Apache Software Foundation are elected by a group of approximately 100 most active technical contributors (Apache, 2004).

Finally, governance must explicitly incorporate the concerns of the central stakeholder, the VA. Due to FOIA, the VA cannot block the growth of an open source community. But given its technical knowledge

and continuing investment in the code, few of the project goals could be realized without its ongoing and enthusiastic participation.

3.6 Centrality of Domain Knowledge

Many successful open source projects share at least one of two common characteristics. First, many projects (such as web servers and development tools) are developed by programmers who will be the eventual users; their interest in the project is tightly coupled with their own needs. Second, some key technologies like Linux are infrastructure technologies that are not directly visible to users, but form the basis on which user applications are constructed. The VistA system shares neither of these characteristics.

First, the VistA system depends heavily on expertise from domains outside computer science or related fields such as web protocols. These domains include medicine, pharmacology, accounting and health care regulation, and thus parallels Enterprise Resource Planning systems such as SAP R/3. This means that the ongoing development (and governance) of the project must incorporate the contributions of non-programmer domain experts, in a way that legitimates them as first class citizens of the project.

Since the users of the system are skilled medical professionals, including doctors, nurses and pharmacists, the system must evolve to keep the centrality of their requirements paramount. If VistA is to be used to transfer the VHA technology to private and non-profit hospitals, then the governance of the project must incorporate requirements beyond that of the VHA. Thus project sponsors would want to recruit not just talented programmers, but domain level experts who could contribute programmatic expertise.

3.7 Intellectual Property Licensing

The VistA project has three licensing goals typical of a federally sponsored project:

1. Encourage the use of a unified distribution of VistA code for use outside the VA;
2. Assure that contributions to VistA are widely disseminated to all possible users (including the VA); and
3. Attract a supply of complementary solutions (e.g., new modules) to extend the capabilities of VistA for use by commercial, non-profit and government health care providers.

The reciprocity of the GPL requires users to contribute to a single code base and prevents distribution of proprietary derivative works. If the project uses the GPL, it might encourage greater dissemination of a unified distribution, but discourage provision of complementary solutions. To meet the latter goal, the project could choose the Lesser GPL (LGPL), which allows developers to add new proprietary modules although they could not privatize proprietary enhancements to VistA. This option has allowed major IT vendors to develop applications for Linux, and allows OpenOffice to be used as a building block for various open source and proprietary packages.

However, the VistA Project has a unique licensing challenge as releases of the FOIA version of VistA in the public domain may “trump” a version licensed under more restrictive terms. Given a choice between the unrestricted FOIA VistA and the limits of a copyleft license, firms might prefer to use the FOIA code, which would only continue the existing forking.

This suggests that for cases where a public domain supply is perpetually available, an open source project such as VistA would favor an open source license that does not restrict distribution of proprietary additions to it. The use of a popular non-viral license such as Apache, BSD, or Mozilla would provide a familiar license form that encourages proprietary extensions.

4. Tradeoffs in Community Building

4.1 Challenges for Community-Led Projects

Building community in parallel with the code base offers several advantages. It allows early entrants to lay their fingerprints on the architecture of the project. This can enhance the robustness of the architecture itself, help foster the development of shared formats, processes, and norms; and increase individual commitment to the project.

At the same time, this process is often marked by long periods of ambiguity over project direction and viability and take time to develop critical mass. Furthermore, many of the mature, successful projects were founded well before the popularization of open source software changed the ecology of community developed software. Now, there are more projects competing for quality contributors and more commercial entrants who may make it more difficult to align community and firm interests. Thus, many open source projects do not make it past the incubation stage for some combination of these reasons.

4.2 Challenges for Sponsored Projects

Projects spun-out from proprietary sponsors have several advantages. Their architecture may be more clearly defined, their viability more clearly established, and they may have received commercial and/or community recognition which can help attract talented developers. Sponsored projects have better assurance of the support needed to develop a ‘commercial face’ by marketing and promoting the project.

However, these projects lack several features endemic to community initiated projects. The ability

for contributors to stake out autonomous sections of the project to which they can uniquely contribute may not be clear in the presence of prior "owners" to the code. If contributors do not see how they can "make their mark," they will be less inclined to contribute to a sponsored project since this has been identified as a critical incentive on community managed projects.

Since sponsored projects do not have the opportunity to recruit developers in the design phase of the project, managing the appropriate level of modularity may be more important. A degree of modularity that attracts talented developers seeking to exercise their skills and autonomy will be critical. Potential contributors may react negatively to spinouts that are viewed as transfers to the community to maintain code as opposed to collaborative partnerships. Community based contributors will be quick to note if contributor pluralism is nominally encouraged, yet the governance structure allows a dominant to emerge.

Thus we predict that pre-emptive governance on the part of a sponsor will have a curvilinear relationship with the project's ability to grow an open source community. If the sponsor does too much pre-emptive governance design, potential contributors will question the sponsor's motives and be reluctant to become involved. On the other hand, if the boundary between commercial and community ownership and control is not secured, potential contributors will be less motivated to contribute.

4.3 Learning from VistA

The VistA case is unusual as the code is in the public domain and a distributed community of users and developers already exists. Unlike other corporate sponsors, VistA does not have to build an external community from scratch, but must integrate a pre-existing, but fractured community to create a shared code base. Furthermore, they must do so in a way that creates avenues for domain experts to participate and considers the needs of a large group of public and private stakeholders.

The VistA analysis suggests that both public and private sponsors need to create some type of buffer institution to manage the community-commercial boundary. The tensions involved in maintaining this boundary are not unique to firms. In order for a sponsor of any kind to grow an external community around a previously developed project, a buffer organization is needed to create an appropriate distance from the host.

Without a buffer organization, it will be difficult for the sponsor to gain legitimacy in the eyes of all stakeholders. Providing a buffer organization may also help balance the curvilinear relationship between sponsor control and community building potential. This suggests additional considerations for the future study of sponsored open source projects.

Thus far, our analysis of sponsored open source projects identified four roles for sponsors:

providing code for a working system;
providing resources, such to support hardware, web hosting, conferences, travel, marketing, ongoing programmer time or legal counsel;
transferring knowledge regarding code, its history, design and structure;
providing community leadership, to provide a clear mission, allocation of rights and responsibilities, and a structure for community participation and governance.

These roles may be temporary, and sponsors may shift their attention among these roles depending on the maturity of the code base and the degree to which there is a pre-existing community.

The example of VistA adds another possible role to this list: helping create institutional boundaries as well as bridges among community, public and private interests. We predict that this role will be particularly important for sponsors that lack legitimacy in the programming community.

4.4 Measuring Success of a Sponsored Project

How should success of a sponsored project be measured? In addition to measures of a project's internal health (such as achieving pluralism), the project should be evaluated by its alignment with the sponsor's open source strategy.

We can suggest three reasons why sponsors would initiate an open source project. First, they want to create a larger market for the project's software or reduce a competitor's market share by building a robust development ecology. Second, the use of external contributors reduces the demand on the sponsor's resources so they can be redeployed. Third, the sponsor hopes to create software as a public good – particularly likely for a nonprofit or government sponsor.

No one measure of success matches all strategies. Sponsors trying to accelerate the project's market share will be more interested in the rate of membership growth, rate of adoption, strengthening of their brand, and innovations that complement the project. Those seeking cost reductions will concentrate on indicators of a self sustaining project, such as the diversity of project contributors and the project's ability to attract resources from others. Sponsors creating a public good may focus on the degree of awareness and adoption of the project.

5. Implications

This paper has identified key distinctions between community and sponsored open source projects. Our research suggests that firms experimenting with the open source model are more likely to do so in an

adaptive as opposed to a wholesale fashion, but how these hybrids will survive is unknown.

Creating a spinout open source project shares many of the same issues as a community managed one, such as building a collaboration infrastructure, designing governance mechanisms and making key decisions about licensing and other external relationships.

Any ongoing relationship between the sponsor and the community will raise questions of ownership, decision rights, and control. West (2003) predicts an "essential tension" of open source projects: trading off appropriation of the returns versus providing incentives for adoption. In this case, a sponsor's desires to guide the project to meet its own needs (and establish licensing terms favorable to its own interests) must be weighed against providing incentives for participants to join and contribute to the community.

To the degree that sponsors assert unilateral ongoing technical leadership and control, sponsored projects may have difficulty recruiting external contributors. Since external participation in the pre-spinout technical design phase is not feasible, it will be more critical for sponsors to seek external participation in the design of governance mechanisms. When doing so, sponsors must walk a fine line between asserting preferential decision rights and losing all influence over the project. A well functioning governance model should enable the sponsor to influence the project's evolution through pluralistic support.

5.1 Technology Transfer

Converting federally funded software to open source software could allow pre-existing investments in software to further economic development and benefit the sponsor. The success of using open source as a technology transfer mechanism will vary based on a number of factors, including:

Government priorities. As with private firms, some government spinout projects will represent an exit strategy for a technology that otherwise lacks ongoing value. In these cases, the government will be less likely to want to control the community but also less likely to provide ongoing resources to support the community.

Applicability of technology to external needs. Obviously some technologies will be more transferable than others to the requirements of other jurisdictions or private entities.

Financial goals and constraints. Some governments (such as public universities) will seek to profit from their technology. Others will seek to discriminate in favor of local or national champions to control the spillovers. These goals are more consistent with building a "gated community" than a full open source project.

5.2 Future Research

Our study highlights the general lack of empirical research on sponsored open source projects. As open source software becomes not only accepted by the computing industry but a model to adapt and modify in conjunction with traditional corporate strategy, more attention needs to be paid to the evolution of technical and social hybrids that are emerging.

It is likely that the role of spinout projects in a firm's strategy will affect how firms approach community building. We would expect that the degree to which a firm's complementary work can be decoupled from a community project, the more likely they will pursue an open approach to creating a spinout project. For example, firms interested in testing prototype ideas may be more open in their approach to community building than firms building tightly coupled complements.

Thus, points of interdependence will be worth examining in order to understand how much loss of control a sponsor can tolerate. Temporal conditions may matter as well. Longer product development cycles may be able to tolerate more community control, where as shorter product development cycles may require more sponsor control.

Future research should also attend to the dynamics of sponsored open source projects over time. Is it possible for a sponsored project to become a community managed project? The Mozilla and Jikes projects suggest that sponsored projects can indeed achieve self-governance, a necessary if not sufficient measure for a sustainable community project.

6. References

Apache (2004) "Management - The Apache Software Foundation," Apache Software Foundation, http://www.apache.org/foundation/roles.html

Baldwin, Carliss Y., & Clark, Kim B. (2003) "Does Code Architecture Mitigate Free Riding in the Open Source Development Model?" working paper, Harvard Business School, June 1.

Healy, Kieran and Alan Schussman, (2003) "The Ecology of Open-Source Software Development," working paper, Department of Sociology, University of Arizona, January 29, http://opensource.mit.edu/papers/healyschussman.pdf

Lakhani, Karim and Wolf, Robert G., (2003) "Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects," September, MIT Sloan Working Paper No. 4425-03.

Lerner, J. & Tirole, J., (2002) The Simple Economics of Open Source. *Journal of Industrial Economics* , 52, 197-234.

Marshall, Rick, (2003) "About VistA," WorldVistA, Feb. 17, http://www.worldvista.org/About/VistA/

Medsphere (2003), "History of VistA," Medsphere Systems Corporation, http://www.medsphere.com/development/history.wpl

Moglen, E. (1999) Anarchism Triumphant: Free Software and the Death of Copyright. Saggi, Conferenze E Seminari (38), Roma..

Moody, Glyn, (2001) *Rebel Code: Inside Linux and the Open Source Revolution,* Cambridge, Mass.: Perseus.

O'Mahony, Siobhán, (2002) "The Emergence of a New Commercial Actor: Community Managed Software Projects", unpublished dissertation, Stanford University.

O'Mahony, Siobhán, (Forthcoming) "Non-Profit Foundations and Their Role in Community-Firm Software Collaboration." In Joe Feller, Brian Fitzgerald, Scott Hissam and Karim Lakhani, *eds., Making Sense of the Bazaar: Perspectives on Free and Open Source Software* . Cambridge, Mass: MIT Press.

O'Mahony, Siobhán, and Fabrizio Ferraro, (Forthcoming) "Managing the Boundary of an 'Open' Project." In John Padgett and Walter Powell, eds., *Market Emergence and Transformation* , edited by

OSI, "Licensing," Open Source Initiative, 2003, URL: http://www.opensource.org/licenses/

OSI, "The Open Source Definition," Version 1.9, Open Source Initiative, 2004, URL: http://www.opensource.org/docs/definition.html.

Stallman, R., (1999) "The GNU Operating System and the Free Software Movement." In: DiBona, C., Ockman, S., & Stone, M., eds., *Open Sources* , O'Reilly, Sebastopol, CA.

VA, "VistA Monograph: 2003-2004," Veterans Health Administration, Department of Veterans Affairs, September 2002, URL: http://www.va.gov/vista_monograph/.

Välimäki, Mikko, (2003) "Dual Licensing in Open Source Software Industry," Systemes d'Information et Management, January 2003, http://opensource.mit.edu/papers/valimaki.pdf.

von Krogh, Georg, Sebastian Spaeth and Karim R. Lakhani (2003) "Community, joining, and specialization in open source software innovation: a case study," *Research Policy,* 32, 7, 1217-1241.

West, Joel (2003) "How open is open enough? Melding proprietary and open source platform strategies." *Research Policy,* 32, 7, 1259-1285.

West, Joel and Scott Gallagher, (2004) "Key Challenges of Open Innovation: Lessons from Open Source Software," working paper, May 2004.

West, Joel and Siobhán O'Mahony, (2003), "The VistA Open Source Project," URL: http://www.worldvista.org/worldvista/calendar/2004-06-seattle/VistA-Community-12-2003.pdf

Project	Date Created	Sponsor	Current Control
Mozilla	1998	Netscape	independent
Jikes	1998	IBM	independent
MySQL	1995	MySQL AB	sponsor-controlled
Ximian Evolution	1999	Ximian Group	sponsor-controlled
Darwin	1999	Apple Computer	sponsor-controlled
OpenOffice	2000	Sun Microsystems	sponsor-controlled
Eclipse	2001	IBM	sponsor-controlled
Chandler	2002	Open Source Application Foundation	sponsor-controlled
Beehive	2004	BEA	sponsor-controlled

Table 1: Examples of firm-sponsored open source projects

	Reason for Initiation	Key issues	Contributor Motivation	Control
Community Initiated	Solve a problem Create a "free software" alternative to proprietary solution	Garnering resources Building healthy community, attracting talented developers Distributing software Gaining 'mindshare' with minimal marketing	To make software happen To gain fulfillment To build and learn new skills To solve personal and professional problems	Democratic, transparent, usually meritocracy Some leadership and stratification
Sponsored	Achieve greater adoption of software Get development help on areas that are of low priority for the firm (e.g. special dialects)	Gaining legitimacy Building healthy community, attracting talented contributors Resolving ambiguity about control and ownership	To complete areas that are of high priority for contributors To gain visibility by prospective employers To influence sponsor's alignment with complementary projects	Varies, but usually sponsor retains direct or indirect control

Table 2: Key issues for community-led and sponsored open source projects