2009

The Open Source Software Ecosystem

Charles M. Schweik
University of Massachusetts - Amherst

The Open Source Software "Ecosystem"

Charles M. Schweik

National Center for Digital Government
Center for Public Policy and Administration
Department of Natural Resources Conservation
University of Massachusetts, Amherst

NCDG Working Paper No. 09-002

Submitted November 27, 2009

The National Center for Digital Government is supported by the National Science Foundation under Grant No. 0131923. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Support for "The Open Source Software Ecosystem" was provided by a grant from the U.S. National Science Foundation (NSFIIS 0447623). Any opinions, findings, conclusions, or recommendations expressed in this material are the authors and do not necessarily reflect the views of this agency.

THE OPEN SOURCE SOFTWARE "ECOSYSTEM"

Open source research in the late 1990s and early 2000's described open source development projects as all-volunteer endeavors without the existence of monetary incentives (Chakravarty, Haruvy and Wu, 2007), and relatively recent empirical studies (Ghosh, 2005; Wolf {{243}}) confirm that a sizable percentage of open source developers are indeed volunteers.[1] Open source development projects involving more than one developer were seen to follow a "hacker ethic" (Himanen, 2000; von Hippel and von Krogh, 2003) where individuals freely give away and exchange software they had written so that it could be modified and built upon, with an expectation of reciprocation. An early puzzle, of particular interest to economists, was why people would voluntarily contribute their ideas and time to these projects (Lerner and Tirole {{243}}. We'll focus on these fine-scale behavioral questions in Chapter 3, and will explain that there are clear reasons – such as distance learning, signaling, enjoyment, and "user-driven innovation" based on a need (von Hippel, 2005) – that motivate these volunteers to participate.

Other recent empirical studies of open source (e.g., {{253 David, Waterman and Arora, 2003; 243 Lakhani and Wolf, 2005; 242 Ghosh, 2005}} identify both volunteer and paid developers as participants. As Maxwell (2006: 141) notes: "In an increasing number of cases... the production of open source software is a job, not a volunteer activity." And a report by Ghosh (2006) states that open source "has rapidly shifted from a model driven purely by the developer community and university support to one where a main driver is industry" (p. 17). These firms have developed business models around open source, and either somehow support these projects

1 Ghosh's (2005) survey of FOSS developers found that about 17% were paid employees. Lakani and Wolf's {{243}} survey of 684 FOSS developers found that 40% of their respondents fell into the paid developer category.

(financially) or pay their own employees to participate (Riehle, 2007). And, as we will describe below, government and nonprofit organizations are also now involved. In short, the open source development environment is now a more complicated "ecosystem" (Capek et al., 2005). In the literature that exists on open source as of this writing, there are many studies that focus on various components of this ecosystem, but nothing that we could find provides a fully-comprehensive view.

In this chapter, we have two objectives. The first objective is to introduce the idea of open source projects as a special kind of commons: a "common property regime." To do this, for readers unfamiliar, we review the general "Theory of Goods" literature as well as Yochai Benkler's (2006) concept of open source as "commons-based peer production." We then note that there is variation what the term "commons" means, and we are trying to be more precise about the kind of commons open source projects are. As we hope to make clear, our introduction of "open source projects as common property" is not made to insert new jargon into the open source research dialog, but because common property as a concept provides an important foundation for other points made in this book – specifically issues about the "institutional design" of open source projects (described fully in Chapter 5).

The second objective of this chapter is to provide a broad-scale overview of the open source ecosystem. By this we mean what general organizational types – firms, governments, nonprofits, universities and scientific research organizations – are interested in using, developing, and/or promoting open source technologies and the motivations driving this interest. Through a literature review, we provide such a mapping.

The Theory of Goods

The Theory of Goods provides a useful point of departure. Social scientists recognize four categories of goods: private, public, club and common pool resources (Samuelson, 1954; Ostrom and Ostrom, 1977; Ostrom, Gardner and Walker, 1994). Two characteristics define these categories: subtractability and excludability (Figure 2.1). A good is subtractable if one's use of a singular unit of the good takes away from others' ability to use that good. A good is excludable if it is relatively easy to keep others from using the good.

*** Figure 2.1 about here ***

Private goods are ones that are both subtractable and excludable. We often associate private goods with the concept of private property that underpin free market exchange (Benkler, 2003). Think of almost any product you purchase in a store (e.g., food items, electronics, etc.) and it probably falls under this category. *Public goods* are the opposite; they are neither subtractable nor excludable. Typical examples are the Earth's supply of air, or national defense. One's personal use of these goods does not keep others from using them as well, and, in fact, it is difficult to exclude people from using them. *Club goods* are similar to public goods in that they are not viewed as subtractable resources (although some can get overcrowded, such as having to wait for some currently unavailable workout equipment in a workout club), but they differ in that people can be excluded from using them. Examples are athletic clubs and toll roads. Finally, *common-pool resources* are goods that are subtractable but exclusion is difficult. They are often found in natural resource settings (fisheries, forests, water resources) where the resource is limited and can be extracted and where it is difficult to keep people from accessing the resource

(Ostrom, Gardner and Walker, 1994). Here we are explicitly using the phrase *common pool resource* rather than *common property resource*, which is sometimes incorrectly (or erroneously) used to mean the same thing. Common pool resources characterize the situation we just described (subtractable, difficult to exclude). Common property conveys some form of collective ownership (Rose, 2000).

Open Source Software as a Complex Good

Some goods fall into one and only one of these categories. The laptop computer being used to write this passage is a simple example of what is usually treated as a private good. But other goods are sometimes referred to as complex goods, where they have properties that could place them in multiple categories in Figure 2.1. Forests are an example of a complex good. They are subtractable resources that sometimes are treated as a private good (e.g., a paper company that owns forested land, or a non-industrial forest parcel). But even in these private good situations, forests provide ecosystem services that we all benefit from, such as CO^2 consumption, erosion control, animal habitat, and it is these perspectives which place them into the public good category as well.

Computer software is another example of a complex good, and, for our purposes in this book, it is important to consider the two general forms of software: binary and source code. When a software user gets a program to install, what he or she typically uses is the “binary” or “run-time” code. This is a digital storage format that computers can easily read and interpret, because the program logic is stored in sequences of 1's and 0's. This is not, however, what the programmers who develop open source software read and edit. For readers unfamiliar with this

topic, that human-readable software is called "source code" and is also maintained by the software developers. So in short, at any point in time, there are two versions of the software being managed by the developer(s): (1) the source code and (2) the binary or run-time programs that we use in our computers. The developer (individual or organization) has to make a decision about the policy – the type of good – that will be assigned to the binary version and the source code version. In both cases, they could be assigned to a number of cells in Figure 2.1., making both the binary and source code complex goods. Let us consider in more depth the binary or run-time software first.

Once the developer has created the binary or run-time version of a new software, he or she could treat the binary software as a private good and sell it. Given the non-subtractability characteristic of digital data, the developer needs to develop a proprietary license (and accompanying installation software) with a system to restrict the number of allowable installations, and then sell the binary version of the software physically on CD or DVD and in shrinkwrap boxes (creating a subtractable good situation). Or, the developer(s) could go to the second option, where they treat their binary software as a club good, and use the Internet as the distribution medium, providing access and installation authority to the binary software only to members who have paid to join the club. Lastly, the developer(s) could decide to treat their binary software as a public good, making it freely (as in cost) accessible and downloadable over the Internet -- as has been done in open source settings.[2] In all three of these cases, only the *use* of the software is provided, since the software as a good is distributed in binary or run-time

2 At this juncture we should note that binary software could be considered a complex good because of "positive network effect" properties that occur in instances where digital data need to be shared with others (Liebowitz and Margolis, 1998). In these situations, the value or utility of the software increases as the user community grows. For example, users of binary versions of Microsoft Word or Adobe's Acrobat Writer software have benefited when others have also utilized these tools or comparable products that can read their output storage formats (Word's ".doc" and Adobe's ".pdf"). In fact, these formats have now become defacto document storage standards because of these network effects (we'll return to this issue in the section on Open Standards later in this chapter).

format.

But the developer also has the source code, which is where the binary version came from and is readable not by computers but by human programmers. The developers not only need to decide their "policy" for distributing binary software, but they also need to decide their policy for their source code. And like the binary software case, software source code can also fall into several cells of Table 2.1, making it a complex good as well. Like the binary case, the developers could treat the software source code as a private good, where the development and decision to release new versions of the software are controlled exclusively by one individual or organization, such as a firm, and no one from outside the firm can contribute to the future development of the product. "Traditional" closed source proprietary software fall under this model, where they keep the human readable source – their innovation – under lock and key. But even in the world of open source, there are cases of private goods. Riehle (2007) calls these "Commercial Open Source" and uses the database product MySQL and the company Sun Microsystems as one such example. Employees of the firm decide what goes into new releases of the software source code (we'll return to this in the Business Models section below).

Second, there are examples of software source code that fall into the club good category.[3] In these situations, development contributions for new releases are restricted to a subset of developers who have somehow become members of the club which has access to the source. To become a member, you might contribute programmers from your organization to work on the project or you might support the project financially. Through commitments for joint provision

3 What makes this slightly challenging from a literature review perspective, is that agreed upon terminology has yet to be established. Wheeler (2007) refers to open source clubs as "Community Source." But in our view, this is problematic, because the term "community" is often used to describe the all-volunteer mode of open source development (see for example, Lakhani and Wolf, 2005; O'Mahony, 2003; 2005). And others, such as Matusow (2005), use the term "community" to also capture collaborative efforts between industry and volunteers."

and production, members of the club enjoy key benefits (Cornes and Sandler, 1996), including the use of shared resources and full access to the source code. The Sakai project is one of the more famous open source clubs, which, at least initially, was a collaboration between a number of universities who were contributing programmers or finances to the development of a web-based course management system. In the Sakai case, the source code is a public good (it is freely available), and membership is optional. But the members in the club participate in the governance of the associated nonprofit foundation, and help determine priorities for the project (Sakai, 2008). There are other open source clubs that go further and treat access to the code as a club good. The "shared source" concept described by Matusow (2005) is one such example, where Microsoft is providing access to the source code of some of their products to some customers and business partners.

Finally, software source code as a good can be treated as either a public good or a common pool resource as well. As a public good, it can be served and distributed over the Internet and be accessible to anyone. Most open source software falls in this category. Or, the source code can be distributed on some subtractable resource only, such as a CD or a DVD, hence making it fall in the "common pool" category in Table 2.1. Both of these are sometimes referred to as "commons" situations (Bollier, 2002; 2008; Goldman and Gabriel, 2005; Benkler, 2005; 2006; see Figure 2.1 dotted box). However, before we discuss open source production as a "commons," we should clarify the meaning of the term.

Open Source as "Commons-Based Peer-Production" and "Common Property"

The term "commons" has been used to describe a wide spectrum of goods (natural or

human-made) that groups of people use, need or rely upon. By definition, commons involve collective action, where two or more people work collaboratively toward a certain outcome (Sandler, 1992: 1). Over the last twenty or more years, there has been a significant body of interdisciplinary literature on the commons produced (see Hess, 2007 for a comprehensive bibliographic database). Much of the focus has been on natural resource commons (forests, irrigation systems, fisheries, the Earth's atmosphere, etc.), but more recently a deep concern has emerged over information and knowledge as a commons (see for example, Boyle, 2003; Kranich, 2004; Hess and Ostrom, 2007). While the term "commons" is widely used with a variety of meanings (Hess and Ostrom, 2003), it is usually associated with the provision of public goods (Olson, 1965; von Hippel and von Krogh, 2003). Moreover, commons usually have established sets of institutional arrangements governing the rights to access, use and management of the good (Dietz et al., 2000: 18; Benkler, 2003: 6; 2006: 60). (We'll return to these ideas in Chapter 4).

Yochai Benkler (2003, 2005, 2006) has worked carefully to describe commons in information production situations (see, in particular, 2006: 61-62). In his seminal work, *The Wealth of Networks*, he coins the phrase "commons-based peer-production" (2006: 63), to describe special circumstances where no centralization exists, no hierarchical assignments occur,[4] and individuals self-select what to work on. In the best of circumstances, large numbers of individuals are involved who are scouring their environment in search of the right project and project component to work on (Benkler, 2005: 178). Benkler refers to open source as the "quintessential instance" of commons-based peer-production. As such, he describes open source this way:

4 This is an interesting and important contradiction to what is known in more traditional commons settings, where governance and institutional design is critical. See for example, Ostrom, 1990; or Dietz, Ostrom and Stern, 2003. We will return to this issue in Chapter 4).

> "...open source, is an approach to software development that is based on shared effort on a nonproprietary model. It depends on many individuals contributing to a common project, with a variety of motivations, and sharing their respective contributions without any single person or entity asserting rights to exclude either from the contributed components or from the resulting whole. In order to avoid having the joint product appropriated by any single party, participants retain copyrights in their contribution, but license them to anyone – participant or stranger – on a model that combines a universal license to use the materials with licensing constraints that make it difficult, if not impossible, for any single contributor or third party to appropriate the project." (2006: 63).

As we see it, one can look at Benkler's concept of open source peer-production commons from two perspectives: (1) the broad-scale, perspective of all open source projects being worked on across the globe, and (2) the fine-scale perspective of any particular individual project. From the broad, global perspective, the Internet is filled with a large (perhaps massive) number of potential programmers who might be interested in working on any particular open source project. Assuming they want to devote the time (Chapter 3 discusses these issues), they might self-select and either start or join in on a project that suits their fancy. From this broad perspective there is no centralized or hierarchical direction driving the choice of project to participate on (except maybe in the case where firms or organizations are paying employees to contribute to particular projects that are needed to serve the organizational mission).

However, focusing attention to individual fine-scale, individual open source projects, it is important to point out that any individual open source project is a special type of "peer production commons" – what we would refer to as "peer-production common property" (Schweik, 2005). Some readers may feel this adds unnecessary jargon or semantics to the open source research conversation. However, people who study environmental commons (such as the lead author of this book) know that there is significant variation in the term "commons." For example, in perhaps the seminal "commons" article, "The Tragedy of the Commons," Garrett

Hardin (1965) was referring to a situation where the commons, a pasture, was "open access." Common property situations are still commons but ones where property rights exist and the resource is shared. The emphasis on property suggests that there are deeper questions about social relations as well as the management and governance structure (Ostrom, 1990; McKean and Ostrom, 1995).

There are several reasons why we think the more precise articulation of open source as "peer-production common property" is important. The first reason relates to the issue or question of authority in these projects (O'Neil, 2008). Empirical studies of open source projects have identified, at the very least, some loose forms of coordination in open source projects. One oft-cited description of this coordination is called the "onion model" (Nakakoji, et al., 2002; Mikkonen, Vadén, and Vainio, 2007). At the center of the onion is a "project leader" or maintainer who makes some of the decisions about the development of the software. This leader can be the person who initiated the project, or someone who has taken over this role for someone else. Assisting this project leader is a group of "core developers" who sometimes have more authority than other project developers in terms of write access to code libraries, tend to be responsible for some sub-section or set of modules related to the project, and participate more closely with the leader in project coordination issues (256, Jorgensen, 2005; Holck and Jorgensen, 2005; Riehle, 2007). These core developers tend to be "promoted" into these positions of authority based on merit (e.g., their earlier contributions to the project, and the expertise they have demonstrated through repeated interactions on the project), leading open source projects to be referred to as "meritocracies" (Deek and McHugh, 2008; Apache, 2008b; O'Neil, 2008). Beyond this there may be other active developers, who make code contributions at varying levels. Further outside are "peripheral developers," developers who work solely on

fixing bugs, and code readers, who may not fix code but help to diagnose problems. Finally, there are active users who report bugs and possibly contribute to testing and project documentation, and passive users who simply use the software. The essential point is this: open source projects tend to have at least some loose form of organizational structure, where certain participants (the leader(s) or core developers) have more rights and authority over the code than others (O'Neil, 2008). And there have been studies of open source (such as the one just cited – Nakakoji, *et al*. 2002; or O'Mahony 2007 or O'Mahony and Ferraro, 2007) that suggest there is variation in the governance and management of these projects and asymmetry between project developer rights.

The second reason for the more precise labeling of open source projects as "peer-production common property" has to do with the licensing and embedded property rights. Contrary to what some might believe, open source projects do, in fact, involve property rights (copyright) and ownership issues (McGowan, 2001), and this relates back to the previous discussion about project leaders and committers. Raymond (cited by McGowan) defines owner(s) of an open source project as ones who have "exclusive right, recognized by the community at large, to redistribute modified versions" (McGowan, 2001: 24]. According to Raymond, a developer becomes the owner of a project by either:

- Being the person or group who initially started the project;
- Being someone who has been granted authority from the original owner to take the leadership role in future maintenance and development efforts;
- Being a person who takes over a project that is widely recognized as abandoned and has made a legitimate effort to locate the original author(s) to get permission to take over ownership.

McGowan (2001) adds another option:

- "Forking" the project into a second, competing project.

This last option, "forking," as many readers will understand, is a kind of mutiny. The threat of it helps to keep a project owner or leader's power in check (O'Neil, 2008). Forking tends to occur in situations where a project is deemed to be moving technically or functionally in the wrong direction by some of the participants and where there are irreconcilable differences of opinions between developers (Narduzzo and Rossi, 2005). Forking is viable action because of the permission specified in the license that allows for "new derivatives" to be created. The result is a new, competing open source project built upon the same foundational source code (McGowan, 2001). The Joomala content management system (CMS) is an example of a forked project, beginning from the source code of another CMS called Mambo (NOSI, 2007). But even in forked projects, someone new takes on the leadership/ownership authorities in the newly established project.

The third reason for labeling open source projects as common property regimes relates again to leadership and governance issues, and the more complex ecosystem of open source that is moving partially away from the all-volunteer situation to a more complicated landscape of projects that are comprised of volunteers, or paid professionals, or both. As we will describe in more detail in the next section, organizations – especially firms, but certainly some government agencies and nonprofits as well – are now contributing resources (employees or financing) to supporting open source projects. With these contributions, interesting questions arise related to project governance that are, in some instances, very new and still in flux (see for example, Sakai, 2008 as an example of this emerging type of governance).[5] Related to the onion model discussed above, Riehle (2007) emphasized the potential importance for firms in having employees in a position of committer – a person in an open source project who has write access to the project

5 See also Shah (2006). He provides a related discussion on empirical differences between some open source commons settings and commercial open source settings (what he calls "gated source").

code libraries -- on a key open source project in part because they can better align the company strategy to the project and vise versa.

In short, we argue that these three reasons – (1) the existence of some authority (the onion model); (2) the embedded property rights contained within the license, with some recognized ownership; and (3) the asymmetries that exist between owners/leaders, committers and other participants – all suggest that open source projects are a special type of commons – "peer-production common property."

To some readers this discussion may seem technical and, perhaps, trivial. Why should we care whether we call open source projects "peer-production commons" or the more precise "peer-production common-property"? In our view, either will work, but the awareness of common property *focuses critical attention on the evolution and emergence of governance structures and institutional design* related to open source project management (O'Mahony and Ferraro, 2007). As Weber (2004: 189) notes: "The open source process is an ongoing experiment. It is testing an imperfect mix of leadership, informal coordination mechanisms, implicit and explicit norms, along with some formal governance structures that are evolving and doing so at a rate that has been sufficient to hold surprisingly complex systems together." With the emergence of business, government and nonprofit interest in open source as a collaborative model, discussed in the next section in depth, we expect open source institutional designs to only get more complex (see for example Sakai, 2008 as an example of this point). Let us now turn to a discussion of why these types of organizations are getting involved, which, we expect, is resulting in a more complex institutional environment for open source collaboration.

Organization Types and their Motivations for Utilizing, Developing or Promoting Open Source Technologies

Over the last five years or so, it has been clear that private firms, governments and nonprofits are becoming more entrenched and therefore complicating what for a time was considered an all-volunteer open source ecosystem. For example, in the FOSS-US survey the authors report: "Only 8.0% of respondents who worked on projects over 10 years ago had some form of external support. The number rose to 15.0% between 6-10 years ago, 35.4% between 3-5 years ago, and within the past 2 years, 53.9% of respondents have had external support" (David, Waterman and Arora, 2003: 23). As we stated in the previous section, this is important in that it is likely to be creating institutional complexity within and across open source projects.

We'll discuss the institutional issues in further depth in Chapter 5. Before we do that, in an effort to fully describe the open source ecosystem, it is important for us to consider why its composition is moving at least partially away from an "all volunteer" situation, to one where organizations from these sectors are interested and potentially diverting their own resources (employees or financing) to open source projects. In this section we'll briefly review four general categories of open source interest groups – businesses, governments, nonprofit organizations and academic and scientific researchers – and try to explain why they act as consumers, producers and/or champions of open source software products.

Business Motivations

There are two main motivations for businesses to utilize, promote and/or produce open source technologies: (1) to save money in the costs of "non-differentiating" information technologies (IT), (2) to differentiate and grow their business, and, in at least one major case, (3)

to support open source projects, higher education, and, in the process, build a positive reputation and contribute to the public good.

Business Motivation #1: Non-differentiating cost savings in IT

One obvious way businesses have become involved in open source is as users of the technology to support their missions. Perens (2005) uses the term "enabling technology" to describe technology that supports the functions of doing business, and is an embedded cost required to help the organization do business. He argues further that there are two types of enabling technologies: differentiating and non-differentiating. When a company creates a technological innovation that makes their business look more desirable than that of competitor's, this is differentiating technology. Amazon.com's use of "other book" recommendations, and Google's underlying PageRank algorithm are examples of this kind of technology. These kinds of innovations are naturally something the business would want to keep secret, or closed-source.

But Perens (2005) estimates that as much as 90% of the software used in any business is non-differentiating, meaning it is needed to run their business but every business needs it. Many of these technologies are infrastructure-related (operating systems, web servers, databases, etc.) and provide the foundation for developing business-differentiating technologies. To make a business more desirable to customers, Perens argues, firms should be spending more on differentiating software and less on software that does not differentiate their business. Open source helps this by distributing cost and risk across multiple collaborating firms. "The companies that join open source collaborations are seeking to use the software in a non-differentiating cost-center role. *It's not important to these companies that Open Source does not in itself produce a profit*" (Perens, 2005; his emphasis).

Following this logic, firms employ information technology professionals to deploy and manage non-differentiating open source business-enabling technologies (such as the infrastructure technologies described above), and, in this context, may be (perhaps even unwittingly) encouraging paid employees to participate in open source projects. For example, a programmer employed by a firm may develop a new function needed for their business to an open source package, and then contribute it back to the project as required by the associated license. A recent study by the research firm Gartner supports these contentions and predicts that more than 90% of businesses will be employing some sort of open source technology, sometimes in embedded or stealth forms, by the year 2012 (Judge, 2008).

Business Motivation #2: To differentiate and grow the business

In addition to the motivation to use open source technologies to support their "cost centers," businesses are now using open source as a business strategy, or in other words, to differentiate and grow their business. Recent studies by Krishamurthy (2005), Riehle (2007) and Deek and McHugh (2008: 272-279) describe various business models that are associated with open source technologies. Each of these articles had their own unique insights and perspectives. In this section, we briefly summarize six different open source-based business strategies based on these cumulative readings.

1. System Integrators. A system integrator is a firm contracted to provide a client with an "end-to-end" integrated product to solve some customer problem or need. These businesses sell a complete product "stack" containing hardware, software and support services. On the software end, it could be entirely open source, or a combination of proprietary and open source packages. Riehle (2007) notes that system integrators have the potential to gain the most (compared to

other business models) from the use of open source software for two reasons: (1) they increase their profits through direct cost savings by using software that they don't have to pay for; and (2) they can potentially reach a larger number of customers through "improved pricing flexibility" (p. 26). The internal cost savings allows the firm to potentially reduce the price of integrated services to the client, thereby increasing the number of customers who might be willing to contract for these services.

2. Open Source Service Providers. There are two types within this business category: first-level support and implementation-service providers and second-level support, training and development services (Riehle, 2007:29). In some respects, the first-level service providers are similar to the system integrator model above in that they help a client get an open source product installed and operational. But first-level service providers differ from the system integrator model in that they typically focus on open source products only and typically do not offer hardware products. Second-level open source service providers either train clients on open source technologies so they can implement the technologies themselves, or act as a problem-solver on technical issues that are beyond the abilities of the technical staff in the client's organization.

Deek and McHugh (2008: 274) offer an example of a firm – GBDirect in the U.K. (GBDirect, 2008) – that provides both first- and second-tier services. Another example is Limeservice.com, a company that provides a hosting site for online surveys based on the open source software "Limesurvey" which is what we used for our survey research in Part IV of this book. According to its website, the lead developer of the survey software is the owner of the service firm, and some of the profits from the online survey service helps to support further development of the Limesurvey software (Limesurvey.com, 2009). This is an example of a

combination service provider business model with the "Software Development" model (described below). Riehle (2007) notes that one of the keys to a service-provider business model is the ability to recruit and retain high quality people, and this relates to discussions we will have in the next chapter related to open source developer motivations and the idea of "signaling skills."

3. Distributors. This was one of the first business models in the "early" days when firms began entering the open source ecosystem. Distributors compile and sell packaged distributions of open source software as well as support and training services around these distributions. The most widely recognized "field" of distribution is in the area of Linux (the operating system). There are a number of different Linux distribution companies – Red Hat (Fedora Linux), Novell (SUSE Linux), Canonical Ltd. (Ubuntu Linux), Mandriva (Mandriva Linux) and others. There are many components that could go into a distribution, such as the latest stable version of Linux, along with various complementary products such as the most recent web server (e.g., Apache), driver software for input and output, desktop environments (such as KDE or GNOME), and other associated open source packages. A benefit for many end users of a Linux Distribution is that they do not need to become a highly technical expert in Linux to install and set up the system, and some distribution companies offer multi-year contracts where the end user will receive the latest security patches, as well as enhanced administration tools (Deek and McHugh, 2008).

4. Hybrid Proprietary/Open Model – Horizontal Development. Horizontal business models describe the situation where a business is not directly dependent on an open source product, but rather benefits from its existence and wants to support it. The "hybrid proprietary/open" component of this label is meant to convey that horizontal approaches can be used in an "all open source" or "pure open source" strategy (open model) or as a strategy where

both proprietary software and open source are utilized together. Deek and McHugh (2008: 277) note that open source provides low cost software to run on their equipment, which potentially improves the sale of their equipment. In the software industry, some firms may wish to support an open source project (either with programming resources or financially), as part of a strategy to "take on" a competitor. IBM is perhaps the most well-known company taking this strategy (as well as the Systems Integrator strategy). They devote resources to projects that are seen as "functionally connected" to some of their own important products (Capek et al., 2005). IBM's support of Linux provides an example of this strategy in competing with Microsoft over operating system dominance. "To the extent that Linux erodes Windows market share, IBM is strengthened because its products run on Linux, while Microsoft is simultaneously weakened..." (Deek and McHugh, 2008: 279).

5. Hybrid Proprietary/Open Model – Vertical Development. Vertical development is meant to convey the process of combining various open source software packages in new ways toward new products or services and selling this new product. Deek and McHugh (2008: 276) use the companies Yahoo and Google as examples of this strategy. These companies have created proprietary products and services that are built upon an open source infrastructure. Google, for example, uses the Linux core operating system (rather than a distribution) to drive its internal servers, and other open source products such as the open source programming language Python and the database package MySQL. But in neither instance – Google or Yahoo – is their internal code shared. Rather they sell these innovations to other customers. According to the author of the PhP programming language, Rasmus Lerdorf, this vertical business model provides the most lucrative opportunities for the future (Deek and McHugh, 2008: 276).

6. Software Development Firms. We've already mentioned one software development

model (Limesurvey, above) where the development of code is supported by providing services for using that code. In that case the firm relies on service income to continue supporting further software development. However, this category is meant to capture a "software producer" who either: (1) combines the source code from an already existing product with some other code they produced themselves to create a new software product; (2) takes an open source product and "bundles" it with other existing but stand-alone modules to create some new suite of packages for use by an end user (Krishamurthy, 2005: 283); (3) decides to re-license proprietary code under an open source license instead; or (4) decides to "dual license" their software.

Related to the first two options above – (1) combining code, or (2) bundling – Krishamurthy (2005) notes that there are two types of software development business models in this subcategory, and they are distinguished by the general type of open source license used (we'll talk about licensing more in Chapter 5) – the GPL model and the non-GPL model. For readers unfamiliar with these, the General Public License, or GPL[6], is the "granddaddy" of all open source licenses, and the brainchild of the famous (in the open source world) programmer Richard Stallman. As we noted in Chapter 1, it was Stallman's innovation in creating the GPL license that really started the open source "movement."

By licensing a piece of software under the GPL, the developer ensures that the software is kept permanently open and free (as in freedom), meaning that it can be used without restriction, the source code is accessible, it can be modified, and can be redistributed with modifications falling under the same GPL conditions. This last requirement – modifications and the reissue of GPL licensing – is often referred to as the "copyleft" innovation. It mandates reciprocity (Rosen, 2005), whereby the recipient of GPL-licensed code is required, upon redistribution of any new

6 For an excellent summary of the GPL and issues related to it, see Deek and McHugh (2008, 250-261).

derivative, to pass on the same stipulated freedoms, by licensing this new derivative under the GPL as well (Krishnamurthy, 2005; Deek and McHugh, 2008).

Since the introduction of the GPL by Stallman, many other open source licenses have been developed (see for example, Rosen, 2005), some of which have looser requirements or are more permissive than the GPL – what we call, in later chapters, "GPL-incompatible" licenses. In these GPL-incompatible licensed circumstances, firms can derive new works from open source software but are not obliged to license new derivatives with the same license. In these instances, firms can develop new derivatives but are not obligated to expose the code logic in this new product (Krishamurthy, 2005).

This distinction between GPL'd open source and GPL-incompatible licenses creates two different type of software development business models (Krishamurthy, 2005), and the difference is whether the source code of the derived product is open access or not. In non-GPL software-development business models, the software-development firm uses open source code to develop a new derivative which is licensed differently than its parent code, and the firm may have saved development costs by using the parent open source code as an input. In GPL software-development models, the firm is required to make the new derivative code available under the GPL as well. But in some instances, this may create tighter bonds between the development firm and customers, through code openness (Krishamurthy, 2005). The client may get the source for free (as in cost), but may pay the developer to create the new derivative, and create new service opportunities for the developer.

The third software-development option for businesses, in addition to combining or "bundling" software, is what West and O'Mahony (2005) call a sponsored "spinout." In these instances, a proprietary software-development firm decides to open up its code and release it

under an open source license, inviting the external community to join in on the project in an effort to increase adoption of the software, or to gain development assistance. This could be undertaken by a firm, government agency or a nonprofit which has developed code, but as far as we can tell, appears to have occurred more often in private sector settings. Examples of this are Netscape forming the Mozilla project, and IBM releasing its Java compiler to form the Jikes project (West and O'Mahony, 2005: 3) or its Eclipse project (O'Mahony, 2007).

Dual licensing is the fourth option for software development firms. Under the dual licensing business model, the firm offers two versions of their software: an open source licensed version and a commercially licensed version. Deek and McHugh (2008) explain that there are two general dual licensing strategies: (1) "enhanced functionality," exemplified by the company Sendmail, Inc.; and (2) the "same functionality" approach, exemplified by the firm MySQL AB.

Sendmail Inc. provides two versions of their "Sendmail" software which transfers email from one host server to another. The open source version provides, essentially, "base" functionality. The commercial version provides enhanced functionality, such as an improved installation component, as well as other improved security components including virus checking. Obviously, Sendmail makes money selling its commercial product while at the same time managing the continued development of the open source version.

The firm Sun Microsystems now owns, develops and supports the popular MySQL relational database software. But Sun purchased this from a former company, MySQL AB, recently. According to Deek and McHugh, prior to this purchase, the staff at MySQL AB did all of the development on the MySQL product in-house. That is, they accepted enhancements or patches to their open source version from outside firm participants, but anything that is contributed is usually re-implemented by some of the company's 300+ full-time employees into

the MySQL product (Hyatt, 2006; referenced in Deek and McHugh, 2008: 65). The company makes a profit by giving permission, through the commercial license, to other companies or commercial software developers so that they can integrate the MySQL database application into their own proprietary products and sell the resulting product as a closed-source, proprietary package. Fogel (2007: 156) argues that this model works best in instances where a firm provides "code libraries," rather than standalone applications.

Business Motivation #3: To support open source projects, higher education, and, in the process, build a positive reputation and contribute to the public good

One relatively high profile "outreach" effort by a firm to support open source is Google's "Summer of Code" program. Beginning in 2005 and growing ever since, according to Google, this program is established to (1) inspire young developers to participate in open source development; (2) provide college students in computer science and related fields with paid opportunities in the summer to do work related to their academic interests; (3) to expose these students to the open source, distributed development approach; (4) get more software written; and (5) to help open source projects identify talented new developers (Google, 2009s). Since 2005, the program advertises that it has linked over 2500 student developers with open source projects, most of which do not compete directly with Google's business interests (Google, 2009b). Little appears to be written on this program that describes why Google sponsors this program and provides stipends for three months of code, other than the material on the program website. But clearly, this effort does achieve the objectives listed above, and through this outreach helps to build or maintain a positive reputation of Google with the open source

development community.[7] Other firms have attempted in their own ways to make similar positive connections with open source software developers, and in some cases, utilize this outreach as a way to promote their business interests (thus connecting to Business Motivation #2 above). The support of the Eclipse software development environment, by IBM and others is one such example.

In sum, this section on Business Motivations emphasized three reasons for businesses to utilize, support and/or be a proponent of open source: (1) to reduce costs in internally used non-differentiating IT, (2) to build and grow their business, and (3) to provide some public service and at the same time, perhaps build a positive reputation with the open source development community. Related to the second motivation, we described several business models that appear to be dominant. In each of these types of motivations, there is potential for firms to support open source projects – either through paid employees, direct project financing or some other kind of assistance. As we will see in Chapter 3, some of these business interests motivate programmers directly through pay or indirectly by encouraging volunteers to build and showcase skills.

Government Motivations

Governments have been showing a gradual and growing interest in open source-based solutions. It's fair to say that the strength of the interest is strongest outside the United States, but with a growing interest emerging in the U.S. as well. At least three categories of motivations drive this interest: financial, public good and independence/economic development.

[7] I've witnessed this first hand at the FOSS4Geo conference in Capetown South Africa in 2008, where there was definitely a positive attitude toward Google and their keynote presenter, and the audience. Some open source projects represented in that audience had mentored student developers that were sponsored through this program.

Government Motivation 1: Cost savings in IT

Like their business counterparts, governments have non-differentiating technology needs (Perens, 2005). Especially in periods of tight finances, governments look for ways to cut costs. Open source technologies with their free (as in cost) licenses, are attractive from this perspective. Of course, the "total cost of ownership" of information technology (IT) is not simply the purchase price. Training, installation, maintenance, technical support, and further upgrades add costs down the road. Moreover, in instances where an organization considers a major shift from one software product to another, there are the costs of migration, and of further training and support. Nevertheless, given that many governments – especially national and state governments – have sizable deployments of IT, just avoiding the costs of annual licensing fees can lead to significant cost savings (Muffatto, 2006). According to one recent study (by an open source advocacy organization), a general trend in the United States Federal government is a move toward data center consolidation, and in these situations, there is an opportunity to shift toward, or at least consider, open source technologies (Vandendriessche, 2007; Gross, 2007). The potential cost savings open source may provide is certainly one factor in this decision.

Related to cost savings is the potential for cost-sharing. Open source as a collaborative paradigm potentially provides the opportunity for government agencies to collaborate and share resources (financial or staff) on IT-development projects or to share code. We referred earlier to such a model in the Sakai project (Sakai, 2008), which is a collaborative project that involves and is led by participants employed by some major public universities. Other examples of efforts in the United States to build government open source collaboration or use are the (now defunct) Government Open Code Consortium (Hamel and Schweik, 2009), the Government Open Source Conference (GOSCON.org; in its fourth year), and projects like the Plonegov effort. According

to its website, Plonegov is a cooperative effort between more than 50 European, North and South America and African public organizations with an interest in developing e-government applications for local or regional governments (plonegov.org, 2008).

Government Motivation #2: Achieving Interoperability and Open Standards

We argue that *the* force driving public sector interest in open source, at least in the United States, is not its financial benefits but rather its public good properties. Since about 2003, the debate in the United States has moved from the question of "open source versus proprietary technologies" to the question of "interoperability and open standards." Governments, as servants to the public, have particular information-flow needs and responsibilities that are different than those of the private sector:

First, they face situations where they need to exchange information both within and across agencies, and also across levels of government. Public safety or emergency response are good examples where this need is critical (consider, for example, the issues facing the U.S. government in response to the Hurricane Katrina crisis).

Second, government agencies often need to exchange information with their constituents, in almost any circumstance imaginable: taxation, social services, public safety and health, license renewals – the list goes on and on. In all of these cases it is expected that the vehicle for information exchange will not place undue burdens on the citizen. For example, in the United States, when e-government platforms were being built to allow online citizen-to-government transactions, government agencies still were required to maintain parallel "manual" or "in-person" systems for people lacking Internet access (Fountain, 2001).

Third, government agencies often collect and store private information about the citizens

they interact with (e.g., tax records), and are expected to be careful stewards of this information. Moreover, in some cases related to national security and defense, there is an expectation that government stewards will protect against leaks of sensitive information.

Finally, fourth, as information stewards, they are expected to archive important records, and make sure that in the future these records can be recovered. In the past this was accomplished through paper archives, but records are now becoming increasingly digital. This is a significant challenge facing governments and other organizations today (Waters, 2006).[8]

It turns out that these needs of government – to be able to seamlessly communicate and share digital information, maintain its security, and provide the ability to recover archived data – were the underlying issues during much of the early debate over IT policy and open source in the United States. The ideas of "interoperable" systems built upon "open standards" are seen as the foundations for solutions to these needs (Simon, 2005). "Open standards," in computing, is a phrase used to describe a generally agreed-upon protocol or structure for the sharing or communication of data. TCP/IP, the underlying communication protocol driving Internet transmission, is one well-established open standard for networking. HTML, the foundational language of web pages, is considered an open standard for web-based publishing. Determining if a standard is truly open depends on the answers to several questions, including how open and fair the process was in creating the standard, whether the standard is publicly disclosed, whether the standard contains any proprietary technology and how the standard will be maintained (Maxwell, 2006). Open source technologies often comply with established open standards, or can be made to do so because their code is accessible. Having software grounded upon open data standards increases the likelihood that interoperability can be achieved, both between current software

8 Any person who has used computers for some time can relate to this problem. For example, I would be hard pressed to be able to find a computer with the software that could read my father's old WordStar documents stored on 5 and ¼ inch floppy disks.

technologies and across time (the digital archive issue).

A high profile case that eventually brought clarity to the issues of open source as a "procurement" policy and the question of open standards and interoperability was the controversy with the Commonwealth of Massachusetts and Open Document Format (Shah, Kesan and Kennis, 2007). This began in 2003 when the Massachusetts' Department of Administration and Finance initiated a state IT procurement policy giving preference to open-source software and products that adhere to open standards such as the Extensible Markup Language or XML[9] and Secure Sockets Layer (SSL)[10] (Commonwealth of Massachusetts, 2004). Their goal was to ensure that all new IT initiatives and (retroactively, legacy systems) conform to open standards with the goal of interoperability – allowing various applications to exchange data more readily (Becker, 2003), and to ensure that past and present state information could be shared with constituents following open data standards. This policy immediately generated significant controversy and some protest, especially regarding their efforts to mandate the XML-based Open Document Format (ODF) as a document storage standard (LaMonika, 2005). ODF is a storage format utilized by the desktop word processor "Open Office Writer," and, at the time, was not available as an output format in Microsoft Office's Word application. The initiative later encountered resistance from disability advocates, because applications which store in ODF were not at the time compatible with assistive technologies for the disabled (Sliwa, 2006). At the same time, the effort by Massachusetts (and others) helped encourage Microsoft to develop an ODF compatible "plug-in" to their "Word" wordprocessor. This new technological fix enabled Massachusetts to move toward a "vendor-neutral" policy without losing its main

9 XML is a computer "language" for marking or "tagging" text to give it semantic meaning. See for example, Sánchez-Fernández and Delgado-Kloosh (2002).

10 SSL is a protocol or standard used to secure Internet data transmissions (see for example, Thomas, 2000.)

policy objective, and without the problems of lack of disability access. It allowed the continued use of Microsoft Office Word with the ODF-compliant plug-in rather than a complete overhaul of government office desktops to the Open Office Writer (LaMonika, 2006). This Commonwealth of Massachusetts experience shows that the real public-good issue has to do with questions about interoperability of software and not necessarily open source.

This example also hints at another potential benefit of open standards – the avoidance of "vendor lock-in." Circumstances where technologies use is built around proprietary communication and/or data formats means that the government agency is making a commitment to that vendor alone, and potentially far into the future (Simon, 2005). The adoption of technologies that conform to established open standards helps organizations avoid this situation.

We mentioned earlier that government agencies are expected to provide information security, and this has led some in government to take a look at open source technologies. However, this is an area of some confusion, with some interested parties arguing that open source technologies are less secure than proprietary software because their code is open, and others arguing the opposite. One recent study by the "Federal Open Source Alliance" (an open source government advocacy and education group supported by HP, RedHat and Intel) that surveyed 218 U.S. Federal civilian, Department of Defense, and Intelligence agency information technology decision-makers and found that there is a significant "perception gap" between agencies that use or don't use open source technologies. Decision-makers disagreed over the benefits and challenges that these technologies provide, and the question of security was the biggest area of disagreement. Thirty percent with open source systems deployed cited security as the top benefit for using open source, while forty percent of the respondents who have not implemented open source saw security as the top challenge (Vandendriessche, 2007; Gross,

2007). Boulanger (2005) provides a helpful summary of some of these issues, and part of the debate comes from the "double-edged sword" of openly available code. On the one hand, having the code open and readable allows criminal hackers or "crackers" to read the logic and look for vulnerabilities in that logic. On the other hand, there is empirical evidence that shows that when a security breech occurs, organizations can respond more quickly on their own (rather than waiting for a proprietary vendor) if the source code is available. Moreover, in some instances, the "having more eyes" peer-production commons situation may lead to a quicker solution to close the vulnerability issue. Boulanger's general conclusion is that neither open source nor proprietary have a particular advantage in this regard, and it really is dependent on the particular packages being compared.

Government Motivation #3: Independence and Economic Development

Governments mostly outside of the United States have implemented or are considering IT procurement policies that either mandate or show preferential treatment toward open source-based technologies (Maxwell, 2006). Such efforts began in Singapore, Germany and Brazil (Hahn, 2002), but have expanded globally over the last five years. Since 2004, researchers at the Center for Strategic and International Studies in Washington, D.C. have tracked press and other media reports related to government open source policies and have assigned them into four categories: research and development (government-based initiatives where open source is either researched or used in some manner), mandates (where open source is required), preferences (where open source is encouraged but not mandatory), and advisory (where the use of open source is permitted). Their most recent report, (Lewis, 2007) found 268 open source policy initiatives, worldwide, with 177 of them being approved. Of these approved initiatives, only six

(3.4%) fell in the "mandate" category, 56 (31.6%) were classified as "preferential," another 56 (31.6%) were "advisories," and the other 59 (33.4%) approved entries were research and development efforts. In addition, they found that regional or local-level governments were more likely to approve open source preferential policies, while national levels tended more often to fall into the "advisory" category. Finally, our interpretation of their summary tables show that, in Europe and Asia, policies were split fairly evenly between research and development, preferential and advisory options. However, Latin American initiatives weighed slightly more heavily toward preferential guidelines for government open source acquisition efforts (Lewis, 2007).

Some of the motivations for these initiatives are the same as we described earlier: financial and interoperability issues (see for example, Ghosh et al. 2005: 205). Lower software costs are particularly important in less-developed countries where government income is relatively low (Maxwell, 2006), and, even in some developed countries, cost savings have been strongly promoted (see, for example, the discussion of the UK by Muffatto, 2006). However, following a similar trajectory to the United States, in their last two years of research, Lewis and colleagues are reporting a shift away from a mandate or even preferential treatment of open source to more of an emphasis on technologies (open source or proprietary) that adhere to open standards (Lewis, 2006).

For many countries outside of the United States, there are at least two other motivations driving a potential preference toward open source technologies: (1) achieving more independence from foreign software companies (or an effort to move away from reliance on one dominant software vendor, such as Microsoft – see Aigrain, 2005: 452 or Maxwell, 2006: 170 footnote 101), and, in some circumstances, (2) an effort to build or support a domestic software

development and service industry. The former is very similar to our earlier discussion about avoiding vendor lock-in, but in this case there may be some added motivation against lock-in with non-domestic firms. And the latter – support for domestically developed open source technologies and service companies – is seen by some as a way to help "level the playing field" between smaller domestic software development firms and larger, more entrenched international competition which have advantages due to "network effects" (Simon, 2005; Evans and Reddy, 2003). Supporting domestic software development appears to be a motivation for China in its efforts to promote open source. For example, several past policy initiatives in China have emphasized the conversion from proprietary operating systems to Linux-based operating systems in the government sector. The move toward Linux was seen as "China's most important chance to improve its software industry" (Lewis, 2007: 5).

Some governments, in support of some or perhaps all of the reasons stated above, have not only used government funds to purchase open source-based technologies, but have set aside funds to support open source development efforts, or even to pay their own government employees to develop code themselves. Brazil is one prominent example, where the national government has not only made conversion to open source technologies in government offices a major effort (Muffatto, 2006; Lewis, 2007), but has also encouraged agencies to develop their open source technologies to serve their needs and to drive domestic industry. One example of this is the TerraLib project, an open source Geographic Information System developed in a partnership between the National Institute for Space Research and the Catholic University of Rio de Janeiro. In this instance, the two partners have invested more than 50 person-years of programming effort into TerrLib development, and the Brazilian government continues to support the core development team (Camara, and Fonseca, 2007). According to these authors,

the effort is helping to build a domestic GIS industry in country.

Germany provides another example where the national government is helping to fund open source development (Muffatto, 2006). For example, their Ministry of Economics and Labor helped found the web-based BerliOS "open source center", which was built to help, as a neutral party, mediate different groups (e.g., users, developers, manufacturers) with interests in open source (http://berlios.de/about). Germany has also participated in developing open source software for government use. As early as 1999, the Ministry helped finance an open source encryption project for email systems. More recently, the German national government has helped finance efforts to develop an open source desktop package for use in public administration settings. This resulted in contributions, for example, to the KDE desktop package for the Linux operating system (Muffatto, 2006: 145).

In summary, the discussion in this section on government motivations demonstrates that government organizations, like businesses, have legitimate reasons to use, produce and/or champion open source technologies. However, the actions by these governments to push IT policy in their countries toward preferential treatment of open source have, not surprisingly, received criticism. Hahn (2002) provides one of the first discussions with both sides of the debate represented. The reaction to the ODF policy in Massachusetts is another case in point (Shah, Kesan and Kennis, 2007). Evans and Reddy (2003)[11] address many of the motivations we described above and argue that (1) there is no market failure condition in the software industry requiring government intervention; (2) there is no reason to believe that government policy-makers have the expertise to help "design new and improved software industries"; (3) they are skeptical of the network-effect claims. They generally argue that the IT technicians, not policy-makers, should be deciding on products to use based on standard decision categories, such

11 See also Schmidt and Schnitzer, 2003 who also discuss government open source policy issues.

as technical merit, total cost of ownership, etc. These criteria seem to be what many governments are now shifting to (along with an eye and emphasis on compliance with established open standards). Ghosh (2005) provides arguments on the other side – in favor of government preferential treatment for open source. This debate will undoubtedly continue and will be entwined with the issue of interoperability and open standards and domestic software industry support.

Nonprofit Motivations

Nonprofit organizations are the third broad organizational category which is active in the open source ecosystem, and their motivations for adopting, developing and/or promoting open source technologies is similar to some of the government motivations: (1) potential IT cost savings and, in some instances, (2) an alignment with their public good and collaborative philosophy and culture. However, with the exception of large nonprofits, international nongovernmental organizations, or specialized nonprofit technology organizations (described more below), in many cases, nonprofits may not have the technical expertise or the time and resources to participate to the same degree as businesses and governments.

Nonprofit Motivation #1: Cost savings in IT

A recent survey by the Nonprofit Open Source Institute (NOSI, 2008) reports that open source technologies currently in use by nonprofits are primarily web server technologies (e.g., Apache, MySQL databases, content management systems like Drupal), and desktop applications (e.g., Firefox web browser, Open Office, MySQL) running on proprietary operating systems such as Windows. Few organizations appear to be using the Linux operating system on their

desktops (NOSI, 2008). The prospect of saving money on IT and the idea of being able to freely download and install various applications (servers and desktop applications) motivate nonprofits to use open source. In addition, open source technologies provide opportunities to reuse older computers for firewalls or for low-level office computing needs (McQuillan, 2008).

However, Peizer (2003) has warned nonprofit IT decision-makers to carefully consider the total cost of ownership question. He argued that nonprofits differ from businesses (or, in our view, governments) in that nonprofits can't as easily recover from a poor choice of technology strategy, and that the total cost of open source in many instances may be as high as or higher than comparable proprietary applications. He also notes that most nonprofits do not have the technical staff capable of taking on installation and maintenance of many open source products, which are often more technically challenging than comparable proprietary products. This appears to still be the case. The NOSI study notes that the most significant barriers toward open source adoption were organizational familiarity with proprietary tools, and a lack of in-house open source expertise (NOSI, 2008: 3).

Nonprofits, especially the small ones, are constantly facing the challenge of managing technology with very little or no technical know-how on staff. As someone who teaches mid-career graduate students who work in the nonprofit sector, I have found it common to hear about the “accidental technologist” – the person on staff who seems to know the most about technology because they have experience with it (e.g., the accountant who knows spreadsheets) or a person who likes technology, but was never hired specifically to be in IT support. For this reason, nonprofit technology service providers such as (in no particular order) the Nonprofit Technology Network (nten.org), TechSoup.org, Npower (a network of locally-based nonprofit IT service providers, npower.org), the Association for Progressive Communications (APC.org),

Itrainonline.org, Geekcorps.org, and others, have appeared. These organizations are trying to help close the "technology know-how gap" in general and regardless of whether the focus is open source or proprietary technologies. However, some of these organizations actively promote open source. For example, at the time of this writing, the Nonprofit Technology Network had a number of upcoming web seminars on a variety of open source technologies, such as Community Resource Management, Linux-based desktop, blogging tools, etc. (NTEN, 2008). There are also nonprofits that have emerged specifically to promote the use of open source software (see for example, NOSI, 2008c).

Nonprofit Motivation #2: Align IT with their "open and collaborative" culture

McQuillan (2008) talks about the "overlap of values" between nonprofits and open source communities around the ideals of collaboration and cooperation. The group at the Nonprofit Open Source Initiative identified three specific areas where, philosophically, nonprofits and open source connect. First, nonprofits often rely on collaboration as a foundation toward meeting their mission, so the idea of "community ownership" in software may resonate with some people involved in nonprofit work. Second, the threat that corporate intellectual property presents to "the intellectual or knowledge commons" in general, and the idea that open source is an alternative to that model, is attractive to some. Finally, the nonprofit sector is closely associated or sometimes referred to as the "voluntary sector." The idea that some of these open source applications are developed in a similar volunteer vein is in line with the approach and values of many nonprofits (NOSI, 2007). The recent NOSI survey of nonprofits and open source reported nearly 70 percent of their respondents agreeing that open source was "philosophically or politically in line" with their mission (NOSI, 2008: 3).

Interestingly, the freedom provided in open source – the ability to modify the source – is something that most nonprofits, (except perhaps for very large ones such as large international NGOs, for example), have not taken full advantage of (Peizer, 2003), although there is some awareness of this potential for the future. For example, back in 2002, Boeder (2002) noted that the open source licensing innovation creates the opportunity for nonprofit organizations to share their limited software development resources, as well as the opportunity for application service providers (like the ones listed above) to emerge, who could produce open source applications designed specifically for nonprofits.

Indeed, some development efforts have appeared with a focus on key application needs in the nonprofit sector. Perhaps the dominant examples in this sector are found in higher education, with an emphasis on course management systems such as the already mentioned Sakai project or the "Moodle" platform, research "e-repositories" such as MIT's "Dspace," or other related software applications (Pan and Bonk, 2007). In addition, other open source software to support the management of nonprofits have appeared. For example, a project called "CivicCRM" (CiviCRM, 2008) is an open source "constituent-relationship management" system that allows a nonprofit to manage fundraising efforts, as well as manage and track volunteers, donors, employees, clients, and vendors. Based on the analysis above, CivicCRM could be classified as a common-property project being coordinated by CiviCRM LLC, with its financing going through the nonprofit "Social Source Foundation" (CiviCRM, 2008). For-profit firms are also emerging, following some of the business models described above, to fill nonprofit technology needs. CivicSpace Labs LLC is one such for-profit company, offering a product called CivicSpace. This is an online service to help nonprofits with fundraising and improved online communication with constituents (CivicSpace, 2008). Their business strategy follows the "Vertical Development"

model described earlier, where the firm provides new functions built upon an open source Content Management System called Drupal (Drupal.org).

Academic and Scientific Research Organization Motivations

Academic and/or scientific research is the last group we will discuss who have potential motivations for use, production and promotion of open source technologies. This category can really be considered a combination of the previous three. Academic institutions can be for-profit or non-profit, and much of the research work being conducted is funded through government grants. Consequently, many of the academic and scientific motivations for using, developing and/or promoting open source by preceding groups such as IT cost savings or cost sharing, an open and collaborative IT culture, etc., apply to this group as well.

Academic and Scientific Motivation #1: Cost savings and IT

At our own university, open source technologies are deployed in a number of areas. The open source Apache Web Server supports organization and research websites. The Linux operating system is deployed on many of our university's information technology help organization's computers. Faculty use open source blogging technologies (such as Wordpress) to communicate their thoughts to others and to communicate with students. The open source "Mailman" listserv software is used to support various university email lists. Wikis support various forms of academic communication and collaboration and this is true not only at our institution[12] but elsewhere (Bryant, 2006). These technologies are deployed because they are seen by university IT support staff as robust, trustworthy, they get the job done, and at the same time are help reduce the cost of annual software licenses (Vernon, 2009).

[12] Incidentally, our research team used Mediawiki to coordinate our own research activities.

A recent survey of information technology use in higher education in the United States reports that almost three-fifths of campus CIOs agreed to the statement that "Open Source will play an increasingly important role in our campus IT strategy" (Campus Computing Project, 2007). However, with the exception of Learning Management Systems (such as the Sakai system and a competitor system called "Moodle," which appears to be gaining significant usage), the Campus Computing Project notes that, to date, there are few open source alternatives to support campus administration, such as student information, personnel or finance systems. There have, however, been discussions at the Chief Information Officer and Provost level about the viability for collaboration across universities on these kinds of applications (see, for example, Courant and Griffiths, 2006).

Academic and Scientific Motivation #2: Mandates by funding organizations

Research projects in the academy, often funded by government agencies, have historically contributed significant efforts to the development of open source code (Perens, 2005). Wayner (1999) notes: "The United States government long supported open source software before it became known by that name. Most of the research contracts granted by agencies like the National Science Foundation or the Defense Advanced Research Projects Agency require the recipients to publish the final source code." For example, some of the work on the operating system Unix, was created through the Berkeley System Distribution project funded by the U.S. Department of Defense and implemented at the University of California at Berkeley (Perens, 2005). More recent grant solicitations, such as the U.S. National Science Foundation "Cyberinfrastructure" program, require software products produced as a result of their funding to be licensed open source (U.S. National Science Foundation Office of

Cyberinfrastructure, 2007).

In many instances, the development work supported by these kinds of grants is undertaken by graduate students, working on projects related to their research assistantship or their theses. In the population of open source projects, it may be that a relatively large number of open source software that are available on some public open source hosting sites, such as Sourceforge.net (discussed in depth, beginning in Chapter 6 of this book), were created by college students and related to some component of their academic program. We have been unable to find any empirical research investigating, to what degree this is happening, but it could be sizable.

Academic and Scientific Motivation #3: The need for collaboration to advance a scientific effort, or simply as part of scientific dialog and service to the discipline

It is possible that a significant body of more specialized software, to support scientific research has been made available under open source licenses. As Perens (2005) notes, the community of scientists conducting research in specialized fields may be relatively small. Consequently, there may not be a strong enough profit incentive for firms to create needed analytic software. Open source may be, in these instances, a viable alternative, allowing scholars in very specific fields to collaborate on software tools related to their research interests, advancing the software beyond what any one individual can do alone. An example of this is the Open Bioinformatics Foundation, a volunteer nonprofit organization trying to encourage collaborative open source software development in the field of Bioinformatics (http://www.open-bio.org).

Finally, another motivation driving some academics or other scientists to support or participate in open source is simply because of the desire and incentives in the academy to

contribute or provide service to their discipline or intellectual community. Some authors of software may simply "donate" their innovations to their intellectual community as part of traditional academic discourse and service to their particular field.

The Establishment of Nonprofit Foundations to Support Open Source Projects

Up until now our discussions in this section have focused on the question of motivations of organizations to use, produce or support open source software and projects. But before we move on to other topics it is important to note that a special type of nonprofit organization – nonprofit foundations – have emerged in recent years to protect, manage and market open source projects.

This began back in the mid-1980s with Richard Stallman, when he created the nonprofit Free Software Foundation (FSF) to promote and support Free Software as he and others have defined it, to act as the organization holding the copyrights to various software under GNU General Public License, and to enforce this license (FSF, 2008). More than a decade later, a comparable group called the "Open Source Initiative" (OSI) was formed by Eric Raymond and others to play a similar role in the management of the official definition of "open source" and to maintain a list of "OSI approved" licenses (OSI, 2008). These two groups can be seen as "overarching" nonprofit foundations guiding and promoting free/libre and open source licenses and philosophies.

But, more importantly for our interests in this book, nonprofit foundations have emerged at the project level. The extremely useful work by Siobhan O'Mahony and colleagues (2003, 2005) helps to clarify why this occurred. First, foundations formed to protect developers involved in specific projects. One of the early arguments to establish a nonprofit foundation

connected to an open source project was the protection of developers against potential lawsuits. According to O'Mahony, one of the first examples of this was in 1996-97, with the Debian noncommercial Linux distribution project. There were signs that some might sue over potential patent violations in the Linux code (see the discussion of the SCO Group in Fitzgerald, 2005). In 1997, the nonprofit foundation, "Software in the Public Interest" (SPI), was created to act as the organizational steward for the Debian project. Not long thereafter, other open source foundations emerged taking on similar roles, including the Apache (web server) Software Foundation, the Perl Foundation, the Free BSD Foundation and others (O'Mahony, 2005: 398-399). Since then, the role of foundations has broadened in relationship to projects. Building from O'Mahony (2005: 409), in general, these foundations:

- hold assets for the project (e.g., funds raised, steward of code, hardware such as servers);
- protect developers from liability;
- conduct public relations and marketing for the project;
- facilitate communication or collective action between projects;
- help to diffuse potential conflicts between projects;
- help the project protect their property rights (e.g., enforce the stipulations of the code license).

In addition, foundations quickly became important for another reason: helping to establish or coordinate relationships with firms that had adopted one of the business models stated earlier and that were interested in the project. O'Mahony (2005: 396) describes one of the first examples of this, related to the Apache (web server) project, which occurred before the Apache Foundation was established. In this instance, a Fortune 500 company executive trying to initiate a more formal relationship with the Apache project complained, "How do I make a deal

with a Web page?" By establishing the nonprofit Apache Foundation, there was now an organization that could help broker contractual arrangements and act in the interest of the project. Moreover, O'Mahony (2005) notes that the foundation can potentially help firms:

- donate resources to support the project;
- donate software code and assign copyright to the foundation;
- hire or support individual developers associated with the project; and, in some cases,
- provide mechanisms (such as formal advisory roles) for a firm to help guide the future direction of a project.

These kinds of foundations emerged initially to support a specific project, and in some cases still do (see for example, the Drupal Association (2009)). But a number of these foundations have grown to support not one but a whole suite of projects. These can have some loose connection to one another, or may have no relationship whatsoever other than being affiliated with the same foundation. For example, the nonprofit SPI, discussed earlier, lists 14 associated open source projects, including some well-known ones such as OpenOffice.org (office software suite) and PostgreSQL (database package) (SPI, 2008). The Apache Foundation (Apache, 2008a), often considered a model for other foundations to emulate, lists (as of March 2008) 60 associated projects.

Finally, some nonprofits have emerged specifically to help coordinate efforts between firms and other organizations with some interest in further development of open source software. One of the most prominent cases was the Open Source Development Lab, which in 2007 was merged with the Free Standards Group to form the Linux Foundation (Lohr, 2007). The mission of this foundation is to support key developers and foster the growth of Linux (Linux Foundation, 2008). Another example is the Open Source Software Institute, a nonprofit organization "comprised of corporate, government and academic representatives whose mission

is to promote the development and implementation of open-source software solutions within U.S. federal and state government agencies and academic entities" (OSSI, 2008; Hamel and Schweik, 2009).

Conclusions and Research Questions

We began this chapter with an analysis of open source projects as a "commons" and concluded that in many or perhaps most cases, the projects are more accurately labeled "common property regimes." We made this argument, not to be picky about semantics, but rather to emphasize the important point that because there are property rights involved, these projects have issues of social relations, governance and management. However, for readability purposes in much of this book, we have decided to use the more general term "commons" rather than this more technical phrase. However, readers should be aware that, throughout this book, "open source common property regimes," "open source commons" and "open source projects" mean the same thing. We also emphasized that there are actually two goods managed in open source commons: the binary run-time code, and the source code. We argued that these can sometimes be treated or managed differently.

We then turned to providing a comprehensive, "broad scale" view of the emerging open source software ecosystem. In Figure 2.2 below, we present a schematic summarizing the various components of the ecosystem we have articulated in this chapter. While recent surveys of participants in open source projects continue to show an active volunteer developer community, what we've tried to emphasize in this chapter is that the community composition has been

changing in recent years. The open source ecosystem is becoming more complex, whereby corporations, governments, nonprofit organizations, academic and scientific institutions all have motivations to use, develop and/or promote open source technologies, in addition to individual volunteer developers. We also noted the special case of established nonprofit foundations who protect the interests of individual developers, promote associated projects and to act as a vehicle to make collaboration easier between projects and organizations who wish to support the project.

*** Figure 2.2 about here ***

Our motivation for making both the "common property" argument and also taking the time to describe the motivations of these various organizational types for using, developing and/or promoting open source projects is to show the more complex ecosystem that exists in and around open source. Based on the studies of open source projects so far, the majority of these projects tend to be small teams or small-to-medium sized teams rather than huge development communities, and there are reported asymmetries in the rights of some participants (e.g., the leaders and core committers) compared to others. It is an open question as to what degree business and government interests and foundation oversight boards influence open source governance and management and the rules or constraints developers might be subjected to, but it would not surprise us if asymmetries between team leadership and others are more accentuated in these cases. In these more complex situations where formal organizations are participating either directly (as in the case of some of the business models) or indirectly (such as through foundations), there may be at the very least establishment of some more formal bureaucratic structures. Researchers as early as 2001 were recognizing that governance structure was

potentially important in open source (for example, see Kogut and Metiu, 2001; or Sharma, Suguman and Rajagopalan, 2002). At the same time, it has become clear that governance and institutional designs create "frictions" that developers dislike (Schweik and English, 2005). Weber (2004), Crowston (2005), and Markus (2007) have all emphasized the need for further work on this subject, and scholars such as O'Mahony (2005; 2007) are leading the way. The discussion in this chapter leads to the following research questions presented in Table 2.1.

*** Table 2.1 about here ***

We expect that these more complex open source environments will lead to a more complex governance and institutional setting. But, for the time being, we will postpone making specific statements about testable hypotheses until we discuss governance and institutional design in more detail in Chapter 4. In Part IV of this book, our intentions are to fill in some of the gaps related to understanding the governance, management and institutional designs found in open source commons.

With this broad-scale overview of the open source ecosystem completed, we continue our review of the literature in Chapter 3 to report what is known about the finest-scale component of open source commons: individual developers. Here, we summarize what is known about why they decide, on a day-to-day basis, to continue to contribute to an open source project or decide to "jump ship" and abandon the project.

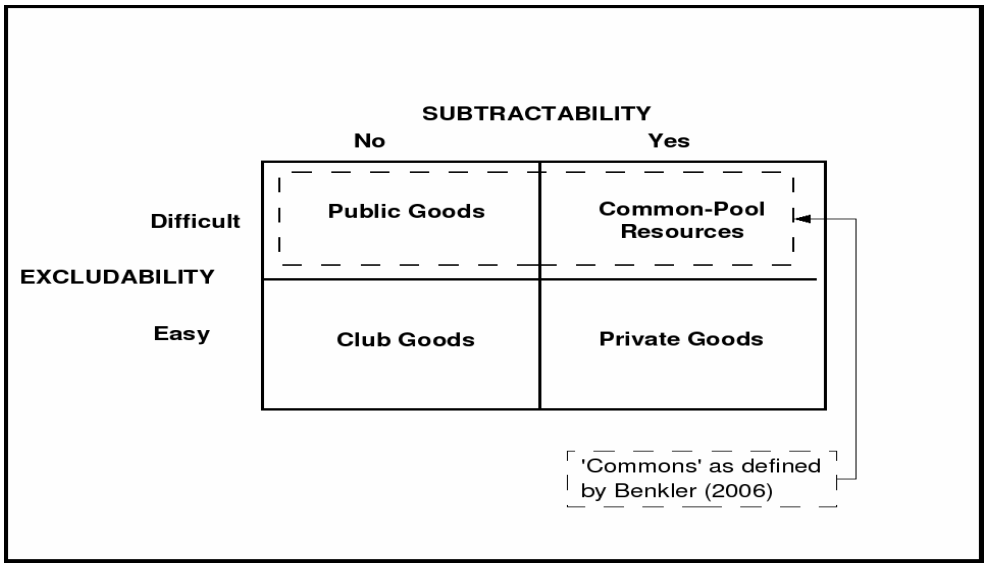

Figure 2.1. The Theory of Goods
(Adapted from Ostrom, Gardner and Walker, 1994: 7)

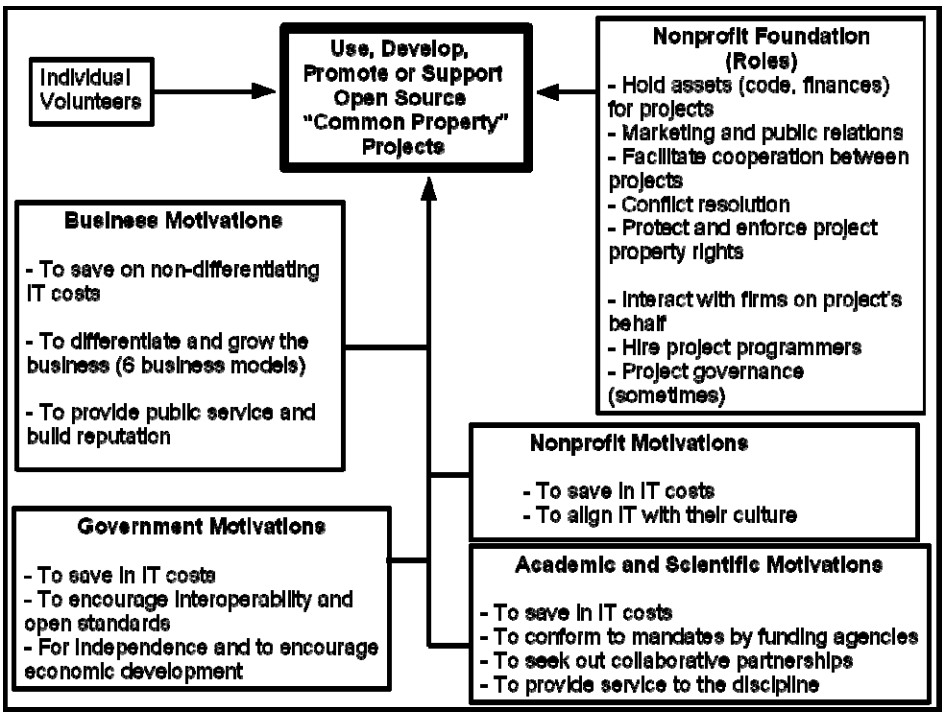

Figure 2.2.
A Broad-Scale View of the Open Source Ecosystem

Table 2.1. Open Source Ecosystem - Research Questions
1. How do open source projects, supported in some way by businesses, governments, or nonprofits, differ from the more "traditional" all-volunteer projects? *2. How does the role of an associated foundation influence the way an open source project is governed and managed?*