First Monday, Volume 8, Number 1 - 6 January 2003

PEER-REVIEWED JOURNAL ON THE INTERNET

The Institutional Design of Open Source Programming: Implications for Addressing Complex Public Policy and Management Problems

by Charles M. Schweik and Andrei Semenov

Abstract

The Institutional Design of Open Source Programming: Implications for Addressing Complex Public Policy and Management Problems by Charles M. Schweik and Andrei Semenov
Recently, an exciting approach to solving complex problems has evolved out of computer science, called Open Source programming. In open source software development settings, programmers freely share their intellectual property — their readable programming source code — over the Internet. Some open source endeavors have resulted in very complex, high-quality software products, of which the best-known are the Linux operating system and the Apache Web server. A great advantage of an Internet-based open source approach is its potential to achieve global collective action toward developing robust solutions to complex programming problems. This paper argues that open source has potential application beyond computer programming. Open source principles could potentially be applied to almost any intellectual endeavor, and may be a very important innovation toward harnessing global collaboration toward solving complex public policy and management problems.

Little research has been published outlining the details of how successful open source programming endeavors are achieved, such as how projects are initiated and organized over time, what rules for participation have been established, and how the methods for maintaining versions of new submissions have been managed. The institutional designs and management of open source projects could be critical for ensuring participants' willingness to collaborate and for recruiting new team members. This paper and the research program it describes, attempts to address this gap. It provides a summary of the "life cycle" of open source programming projects based on existing literature that is largely focused on high-profile open source projects like Linux and Apache Web Server. It then provides interim results from an on-going study of the institutional designs of open source programming projects. It concludes by presenting some examples of non-programming projects that are beginning to apply open source or licensing principles in areas outside of programming and by presenting an example of how these principles might be applied to complex problems beyond programming in the realm of environmental policy and management.

Contents

Introduction

A relatively quiet, but potentially important phenomenon related to human collaboration occurred at the end of the 20th century in the field of computer science. The phenomenon, called open source (OS) software development, has the potential to change, perhaps dramatically, the way humans work together to solve complex problems in general, and specifically in areas of public policy and management.

Probably the most well known OS project is the computer operating system called Linux. Over the course of the 1990s, Linux gained a substantial user community and is now becoming a serious rival to the dominant Microsoft Windows operating system (Economist, 1998). What is unusual about Linux and other OS software (such as the Apache Web Server and others), is: (1) that extremely complex software was designed, built, maintained (Feller and Fitzgerald, 2001); (2) that it continues to be improved primarily by a global team of volunteers collaborating in a virtual community (Rheingold, 1993) over the Internet; and (3) that the OS software is made available to the world at no cost.

There is considerable excitement (see, for example, Bollier, 1999) and perhaps some exaggeration over the promise of OS. Some proponents argue that OS is a fundamental cultural shift in the incentives leading to collaboration: where social status of individual participants is no longer based on what they possess and can trade, but is now based on the quality and significance of what they give away (e.g., Raymond, 1998b; Learmonth, 1997; Edwards, 2001). Others have cautioned that while OS is an interesting phenomenon, the processes of OS have similarities to other long-enduring approaches to collaboration, such as the generic approach to academic research (Bezroukov, 1999a; 1999b). Arguments have also been made that some of the lessons of OS projects are also found in "closed" or proprietary collaborative approaches (Eunice, 1998).

The argument made in this paper is that the OS phenomenon is something we should give serious attention to, and that there are components of it that make it different from more traditional models of collaboration and complex problem solving. Moreover, we make the argument here and elsewhere (Schweik and Grove, 2000) that the concept of OS has the potential to be applied in many areas where people are trying to craft solutions to complex problems facing humanity, and that we should explore opportunities to try to harness its creative power. In fact, this is already beginning to happen in an area that could be referred to as "Open Content." See, for example, Open Music (2002), Open Law (2002), Open CourseWare (2002) and Open Content (2002a).

The central dilemma over OS as a new form of collaboration paradigm is that there has not been enough research on the broader population of these types of projects to determine how they work and what leads to success [1]. While theoretical and empirical studies on OS are starting to emerge (e.g., http://opensource.mit.edu/; Feller and Fitzgerald, 2001; Lakhani et al., 2002; Krisnamurthy, 2002), a significant amount of the literature on the properties of OS is based upon only a few cases — in particular, Linux (Raymond, 1998a; 1998b; Torvalds, 1999; Moody, 2001; Learmonth, 1997; Godfrey and Tu, 2000; Browne, 1998), Apache Web Server (Mockus et al., 2000) and a few others. It is an open question whether the success of Linux and Apache are collective action anomalies that would be difficult to recreate elsewhere or whether there is indeed something new and special about this approach to collaboration that could be applied elsewhere if the right conditions were established. But with the recent book by Feller and Fitzgerald (2001) as an exception, there is little published guidance documenting how OS projects are initiated and managed.

This paper — and the research program it describes — is an important step to providing such guidance. In the first section, we describe the concept of OS and then give a detailed review of what is and what is not known about the lifecycle and composition of OS projects. Second, we address the question: "Is OS really a different approach to solving complex problems from the way usually conduct scientific research?" Our answer is "yes, there are differences" and we provide support for this contention. But OS is not a cure-all. Significant analysis has been devoted to a few highly successful, high-profile OS projects (Eunice, 1998) — and some exaggerated claims made about them — while OS project failures are forgotten. The third section of the paper turns to the need to look more closely at success stories and failures to understand critical success factors. We describe our research strategy for studying institutional designs of OS projects and we present some initial results from the first phase of this study. In the fourth section, we close the paper by providing a look at the potential of OS: in particular, we provide some examples of where OS principles are being applied in contexts outside programming, and describe one vision of the future, in the realm of environmental policy analysis and management.

Open Source Software Collaboration — What We Know and What We Do Not Know

Principles of Open Source

Traditionally, proprietary computer software is shipped by a company or programmers in compiled format, which is readable by the computer but not by humans. Software licenses prohibit the copying or sharing of the software. OS flies in the face of this tradition and the core difference is a shift in the licensing parameters. OS licenses give anyone with access to the software permission to download the program and its readable source code, to copy it, and to freely distribute it with one provision: if any user makes an improvement to the software, he or she is obligated to give the improvement back to the OS community. In other words, "... a user must be able to 'look under the hood,' and be allowed to tune, adapt, or evolve a [software] for his/her personal needs" (Godfrey and Tu, 2000) but these enhancements then also take on the provisions of the OS license. Through this adaptation, over time the software will continue to evolve new capabilities and be less prone to errors (Opensource.org, 2002a).

OS began in the mid-1980s as part of the free software movement popularized by Richard Stallman and the evolution of these ideas during the early distributional policies related to the Linux software project (Perens, 1999) [2]. Today, those behind the Open Source Initiative (Opensource.org, 2002a) manage and promote an official OS Definition (OSD) and provide certification for software and OS distributional licenses (Opensource.org, 2002c). OSD defines a set of nine rights that a software license must provide in order to be certified as open source (Perens, 1999; Feller and Fitzgerald, 2001). The first four rights cover the core OS principles:

1. Free distribution. This allows the user of the software to redistribute it using any method, including giving it away or selling it.
2. Source Code. The program provided must include the readable source code and allow the free distribution of this source code. This allows other programmers to fully understand the logic of the program and gives them the ability to make improvements to the program.
3. Derived works. The license must allow changes to be made by others and allow these changes to be distributed under the same licensing terms as the original software.
4. Integrity of author's source code. This clause requires the work of developers to be represented accurately. Some programmers were concerned that future modifications to their original source might be poorly constructed and hurt their reputation as good programmers. One method of protecting the original author's work and still allowing enhancements is by distributing original works with "patches" that are enhancements.

The remaining rights are included primarily to close loopholes and to prevent misuse of the OS concept (Feller and Fitzgerald, 2001):

5. The license cannot restrict distribution to certain people or groups.
6. The license cannot specify fields of research where a particular piece of software can or cannot be used.
7. The associated rights defined by a license also apply to redistribution of the product. This is to ensure that the product, in the life cycle of new releases, cannot be closed as it is later developed.
8. The license must not be specific to a product. An OS software product cannot be "attached" to a certain software distribution. Feller and Fitzgerald [3] use the example of multiple distributional versions of Linux (Red Hat and Caldera). This provision ensures that a piece of software (e.g., a new module) for Linux cannot be restricted for distribution only in the Red Hat distribution.
9. The license must not contaminate other software. This ensures that the license applies only to the software package it is associated with. If, in a software distribution, there is other separate software included, this other software is not under the same licensing specifications.

There now are thirty OSI-certified licenses, each possessing its own nuances (Opensource.org, 2002d). The important point is that in order for a particular software to be recognized as OS developers need to follow these OS principles and the software license should comply with the OSD standards.

The Lifecycle of OS Programming Projects

OS projects involve three major stages [4]: (1) Project initiation; (2) Going "open"; and (3) Project growth, stability or decline.

Stage 1: Project Initiation

Like any area of endeavor, OS projects are initiated because one or more people realize that there is a computing-related problem or challenge left unfilled, and for one or more reasons, they decide to take it on (Godfrey and Tu, 2000). This describes the history of the operating system Linux, where Linus Torvalds, a Finnish graduate student, decided that a cheap Unix-based PC operating system was needed and set out to build it himself (Learmouth, 1997; Moody, 2001). Motivations, "the kernel," and a modular design are three important components of this stage of an OS project.

Very recently studies have emerged on the motivations of general participants in OS projects (Feller and Fitzgerald, 2001; Lakhani et al., 2002), but it appears that no systematic study has focused specifically on the motivations of OS project initiators. However by analyzing discussions in existing OS literature and building upon the work by Feller and Fitzgerald (2001), we can surmise some of the likely motivations of initiators (Table 1, Column 2). From a technological standpoint, initiators are motivated to meet some personal need (Raymond, 1998a); to work on the leading edge of some technology; to address some software crisis; or to provide intellectual stimulation. Likely socio-political motivations for project initiators include the sheer enjoyment to do the work and an interest in taking on a technical rival (e.g., a large, dominant software company provided motivation in the Linux case). Skill-building and low opportunity costs (e.g., nothing to lose by undertaking the project) are likely economic reasons for initiators to start a programming project.

Table 1: Motivations For Participants In Open Source Software Development Projects.
Adapted from Feller and Fitzgerald, 2001;
*numbers in column 4 show the ranking in the OS motivations survey by Lakhani et al., 2002.

Motivations		Project Initiators	Project Participants (Developers/Users)*
Technological			
	To meeting some personal need	X	3, 5
	To work with the leading edge of technology	X	X
	To address a software crisis	X	X
	To have intellectual stimulation	X	1
	To exploit the efficiency of peer review		6
	To share tedious development tasks (e.g., testing, documentation) with users		6
	To leverage the OS community for R&D		6
	To promote innovation		X
	To ensure the transparency of the application		X
Socio-Political			
	To satisfy an intrinsic motivation to do the work	X	X
	To take on a threat, rival, or monopoly	X	X
	To gratify some ego-driven need		X
	To fulfill some need for a sense of belonging to a community		6
	To fulfill a wish to engage in altruistic work		X

	To help overcome the global software digital divide		X
	To satisfy ideological concerns (e.g., software must be free)		4, 8
	To follow the belief that the approach is a good model for a wider domain (future model for work)		X
	To conform to requirements of the license		9
Economic			
	To build skills; e.g., to improve some component of personal skills such as programming	X	2
	To take advantage of low cost and the opportunity to participate with nothing to lose	X	X
	To gain future job opportunities		7
	To gain future consulting opportunities		7
	To 'strike it rich' through future stock options		X
	To take advantage of investor interest in OSS projects		X
	To show support for the paradigm shift from a commodity industry to a consumer-driven service model		8
	To raise product awareness and strengthen brand name		X
	To take advantage of indirect revenues — selling related products and services		X
	To cut costs (e.g., to create a cheaper platform than proprietary alternatives)		X

The second important component of the initiation stage is the development of an initial product for others to build upon — what we might call the project core, or "kernel." For example, Torvalds developed the kernel of the Linux operating system largely on his own and then, when he felt that it was ready to be shared, he made the kernel source code available on the Internet, and encouraged others to help improve it (Learmouth, 1997).

A third critical component in this stage of OS development is good design and the concept of modularity, "for without it, you cannot have people working in parallel" [5]. Modularity means that the kernel itself and plans for its future development is organized around small, manageable pieces. For example, the Apache Web Server software has over 40 modules and Perl (a general purpose programming language developed in an open source approach) has over 1,000 modules [6]. With a modular design, multiple programmers (perhaps unknown to one another) can be working to build new functions into the same module. This parallel approach is thought to spur innovation and can lead to a rapid development process. Modularity also allows development to continue but avoids a situation where the impact of one person's enhancements to a module leads to problems with the work in some other module. Furthermore, modularity enables the project content manager to keep better control as work progresses and as the product gets more complex (Torvalds, 1999).

Closely linked to modularity is good project design (or at least a well-crafted and articulated vision of where the project is going). "The easiest way to get coordinated behavior from a large, semi-organized mob is to point them at a known target" (Opensource.org, 2002b). A concrete vision and a strategy for the future coupled with a modular structure helps to recruit others into the project.

Stage 2: Going "Open"

The second stage of the OS lifecycle is the point in time when the initiator(s) decide to make the project "open." Going open means that the project founders decide to follow the OSD licensing principles (Opensource.org, 2000c) and they select a particular license for the product (Opensource.org, 2000d). But in addition to licensing, at this juncture, five additional components need to be considered: 1) project/product credibility; 2) adequate communication systems; 3) suitable version control systems; 4) effective recruitment strategies; and, 5) appropriate project governance structures and institutional designs.

Project and Product Credibility. The credibility of the kernel and project is an important issue at the time a project goes "open" (Raymond, 1998a). Raymond and the authors of the now infamous "Halloween Documents" (Opensource.org, 2002b) posit some key criteria for a project to be deemed credible by others: (1) there needs to be at least a handful of enthusiastic "core developers" already interested in the project; (2) the project has "plausible promise," both technically and sociologically (i.e., the kernel can evolve into something very good with a little effort, and that the people in the core developer community are enthusiastic and of high reputation); (3) the project or product is something that will attract interest and is innovative; (4) the project is important and deployable for a (future) large number of developers; and, (5) the right amount of the problem has already been solved before it becomes "open." This last point means that there must be enough of the product developed to provide an adequate framework for coordinating future work. But if too much of the product is completed before going open with it, it has the potential of turning other collaborators who might join in the effort into "testers" — a task many programmers find uninteresting (Opensource.org, 2002a).

Adequate Communication Mechanisms. OS projects utilize the standard Internet tools for communication: e-mail, group e-mail lists, and threaded discussion lists posted on Web sites (Behlendorf, 1999). In this respect the Web has greatly improved the ability to communicate and share documentation and modules. Several Web systems are now available whose purpose is to support collaboration on OS projects. Sourceforge.net, for example, advertises itself as the "largest open source development website" (Sourceforge.net, 2002). This site provides free project hosting services, which include version management, problem tracking, project management, backing-up facilities, and various communication tools such as mailing lists and Web discussion forums. Another Web site undertaking a similar mission with a different design approach is www.freshmeat.net. Other sites, such as www.slashdot.com, www.osdn.com, and www.newsforge.com have emerged in recent years to enhance communication and provide news and information to the general OS community.

Suitable Version Control Systems. At the core of any successful OS project is a system for managing the ongoing development of the product — something referred to as version control systems. In recent years, the most popular of these in the OS community is the Concurrent Versioning System or CVS (Fogel, 1999; Behlendorf, 1999). CVS, an OS product itself, provides team members with methods for downloading the latest version of the OS software and includes functions to manage the history of previous versions of modules, problem tracking, and Web access to archived source code (Feller and Fitzgerald, 2001). Another version control system called "Subversion" is currently under development (see http://subversion.tigris.org).

Effective Recruitment Strategies. Another important concern for a project about to go open is choosing the strategies for announcing the project and for recruiting additional project participants. For OS programming projects, this has become much easier because of the establishment of "central" Web sites for OS project hosting such as surgeforge.net and freshmeat.net. Surgeforge.net provides a "project help wanted" option off of their main menu for people to post requests for participation in non-commercial, volunteer projects.

Appropriate Governance/Institutional Designs. As Bezroukov (1999b) notes: "... in each [OS] project in particular, there are political systems with corresponding and sometimes fuzzy hierarchical structures." We think that these concepts include critical variables that may ensure the success or failure of the project. The governance and institutional designs of OS projects are areas where very little research exists, and these are the areas that this paper and its associated research program are intended to inform.

To analyze the institutional components of OS projects, we utilize an institutional analysis framework outlined by Ostrom et al. (1994). Central to this framework is the concept of "working rules," which define what actions are required, prohibited or permitted and what sanctions are authorized if someone breaks these rules [7]. These could be formally written rules or norms of behavior understood and shared by a community. An important aspect of working rules is that they are followed, at least the majority of the time. Formally written rules that no one pays attention to are not, in this context, considered working rules. We expect that within OS projects there are three levels of working rules as defined by Kiser and Ostrom (1982): Operational, Collective-choice, and Constitutional-choice.

Operational rules direct the daily decisions and actions of actors involved in an OS project. They specify how new module submissions are managed, how new design strategies are defined, when new versions are released, etc. For instance, one of the benefits often cited about OS projects is the concept of parallel development, where multiple parties are working separately on enhancements to the same module. In this kind of development structure, there needs to be some rules as to how module submissions are evaluated, and chosen for the next release. In addition, in some projects, there may be tiers of developers with different levels of authority for the project. For example Bezroukov (1999b) states that in OS projects, "... limiting those who can directly contribute to the kernel to a few most qualified developers (the core team) is a good idea." In these contexts, Operational rules define what actors can or cannot do at certain levels of authority within the project team hierarchy. In the Linux context, qualified developers are referred to as "trusted lieutenants" who are recognized as experts in a particular domain (Moody, 2001; Dafermos, 2001).

Collective-choice rules specify who is eligible to craft or change operational-level rules, and also define the procedures for changing established rules. In some instances, collective-choice rules may be defined by some project leader who has total authority over making and changing operational rules and can do this at any time. For example, in the case of Linux, Torvalds is recognized as the supreme authority (Maclachlan, 1999). Raymond (1998a) mentions that "the leader/coordinator" of OS projects needs to have a talent for recognizing good design ideas from bad ones in an environment of parallel development. Given that Raymond is a recognized leader in the OS movement, his comments suggest that many OS projects may have an established system of collective-choice that is governed by a strong and dominant leader. And Bezroukov (1999a) argues: "Open source may sound democratic, but it isn't. Leaders of the best-known Open Source development efforts often explicitly stated that they function as dictators." But other OS projects may have different sets of collective-choice rules that establish a project oversight body, such as a committee of core developers who, using some kind of voting system, make decisions about the operations of the project and have authority to make changes to established operational rules. For example, in the Apache Web Server project, another prominent OS initiative, the decision-making body is comprised of two-dozen people, all of whom have the right to veto an addition to the software (Maclachlan, 1999).

At the Constitutional-choice level, rules are established to define who is eligible to create or change Collective-choice level rules — a kind of project Constitution, specifying who (or what positions in the team) possess these rights, and outlining procedures for making changes in collective-choice arrangements.

At this juncture we can with some confidence predict that successful OS projects will be ones where collective-choice and operational level rules are at least reasonably well defined, understood, and followed. And we expect projects that go open without some prior consideration about systems for defining and changing operational rules will exhibit problems or perhaps even fail. But it is the more detailed components of the governance of OS projects where substantial variability may lie, and we expect these variables to be important determinants of the success of the endeavor. Hence, the character of these variables is one central focus of our current research into OS projects.

Finally, with respect to OS institutional designs, there are two important areas of operational rules that may be especially important: (1) the system of peer-review and (2) established mechanisms for conflict resolution.

Peer-review is a vital component of OS projects. Similar to the process in academic settings, peer-review is a process where others review submitted enhancements (Maclachlan, 1999). Peer review is a well-understood practice. Less well understood are mechanisms for conflict resolution. In many OS projects, reputation within the group is an important motivation, and in a parallel development approach there may be conflicts between two alternative solutions to a problem. The selection of one solution over another could lead to some potentially serious debates between group members. The worst conflict scenario in OS projects is a kind of intellectual mutiny referred to as "forking." In these instances, members of the group decide to break away from the existing OS project to form a rival project because they do not agree with the design directions the current leadership regime is taking (Lessig, 2001). Because of the licensing rules in OS, they are guaranteed access to the source code and are free to develop it further in whatever way they please. Feller and Fitzgerald [8] note that in some OS projects core developers who have earned substantial respect in the community go on to take on the role of arbitrators. Since conventional project management control is not possible, a supreme leader or acknowledged board of directors is essential in OS projects (Feller and Fitzgerald, 2001). Maclachlan (1999) notes that "dictators" run some of the best-known OS projects with Linux being the ultimate example. We would add that at least some system for conflict resolution is required. In the case of Linux, for example, a "trusted lieutenant" openly disagreed with Torvalds (the recognized leader of the project) about a technical issue and this open discussion led to community support for the argument made by the trusted lieutenant

(Dafermos, 2001). This suggests that there are mechanisms for conflict resolution in successful OS projects — even those run dictatorially — although they may simply be social norms and not formal mechanisms. But how exactly disputes are resolved in OS projects is another area that deserves more attention by the research community, for if they are not resolved in a satisfactory way from the perspective of many participants, this could lead to problems within the group and diminish the likelihood of project success.

Stage 3: Project Growth or Decline

Once a project officially becomes OS, the hope is that it will generate enthusiasm and entice programmers and users in the global Internet community to utilize the existing product and contribute to its further development through programming, testing, or providing documentation. At Stage 3, projects can grow with new membership, remain stable with about the same number of participants as they had before going open, or they can gradually die from a lack of participant interest. The components of Stage 2 — credibility, communication systems, version control, recruitment strategies, and project governance — are the predominant factors in enticing new members to the group.

Some of the motivations for joining an OS project at this stage are the same as for initiating one, but there are some new motivations as well. Lakhani et al. (2002) recently asked some 684 OS programmers to rank their top three motivations for participating in a project and came up with a subset of the reasons similar to the ones identified by Feller and Fitzgerald. In rank order as reported by Lakhani et al. (Table 1, Column 2), computer hackers participate in OS projects because: (1) it is intellectually stimulating; (2) it is helpful in improving their own programming skills; (3) it provides some important product for the context of their job or hobby; (4) they feel it should be open and therefore they contribute some time to it; (5) it provides some important product for their own non-work related interests; (6) it provides sense of community in working with a team; (7) it enhances their professional status (e.g., provides them with an opportunity to do consulting work or other job opportunities); (8) it helps maintain their belief in open source as a 'good thing'; or, (9) it may be simply a response to license requirements that forces them to participate. Feller and Fitzgerald (2001) identify other motivations that are not reported in the Lakhani et al. (2002) survey. These additional motivations are marked with an "X" in Table 1, Column 2.

The rankings in Table 1, Column 2 and additional work by Lakhani and von Hippel (2002) help to emphasize the importance of intellectual stimulation and learning as a motivator for participation. Making the source code available on the Internet provides access to the programming logic of others. New programmers or programmers interested in learning a new technology have the luxury of being able to study the existing program infrastructure — the details of the intellectual property of others in the group — and from this they can participate in a form of distance learning. This is quite similar to the learning that has gone on in the development of Web pages simply because browser technology allows the user to view the Web page source code (Lessig, 2001). In addition, in situations where the programmer submits a contribution back to the community, the peer-review component of the OS process, provides a way to get important feedback on the quality of one's work. As academics know, student independent studies are usually improved when they have an opportunity to discuss what they are doing with another more learned colleague.

The importance of team growth to the success of an OS project is, however, still very much an open question. On the one hand, in his observations on Linux and Fetchmail, Raymond (1998a) emphasizes the importance of large groups in what he terms "Linus' Law": "Given enough eyeballs, all bugs are shallow ones." This function of group size certainly has been a factor in the success of high-profile OS software products like Linux and Apache Web server.

But group size at the "going open" stage of an OS project is a more complex matter than a simple head count. As Cavalier (1998) points out, total size of an OS participant base — i.e., the simple sum of the number of individuals participating — may not be as revealing as what he terms "effective size" and "effective power." "Effective size" is the total number of individuals who actively contribute to a specific activity (e.g., testing) or module development, and more intellectually stimulating areas of the project can have a substantially higher effective size than others. Furthermore, even this measure needs qualification, for the "effective power" of an OS project activity may be calculated as the effective size of that activity multiplied by the average (per-participant) weekly contribution to the activity in hours. These measurements have led Cavalier (1998) to some further observations on group size and effective power. First, some activities will require higher effective size and power than others. Second, if the required effective size of an activity is not achieved, then the activity will likely not be completed. Third, there is often an implicit deadline for project completion (sometimes driven by knowledge of a competing product) and therefore there is some minimum effective power and effective size needed to complete the task on time. Fourth, willingness of people to continue to contribute is related to the progress that is made. If a large number of activities do not seem to be moving forward, participants will lose interest or bicker thus reducing effective size and power. This leads to a higher likelihood of activities not being completed, and ultimately, the death of the project.

Krishnamurthy's (2002) work raises questions about the importance of the group size variable. He reports that out of the most active 100 "mature" OS software products listed on Sourceforge.net, the majority are developed by a small number of individuals (a median of 4 developers and 1 project administrator). He also reports that there is high variation in the number of developers. There have been some that have questioned the study results [9] by arguing that the Sourceforge data used for OS projects is incomplete. But even so, Krishnamurthy's study suggests that although some OS projects achieve significant success with high participation rates, many if not the majority of projects tend to remain in the hands of a small number of interested participants and never achieve the levels anticipated by "Linus' Law." In short, cases like Linux and the Apache Web server may be more anomalies than the norm. Until recently, most literature has been about these high-profile cases, which may be why there are presently some possibly exaggerated claims over OS. Yet clearly, these cases have an impressive ability to generate nearly global collaboration.

OS Lessons and Important Research Questions

To summarize, Table 2 lists key attributes of OS projects and what we have learned from this review of literature on the lifecycle of these projects. In general, OS projects exhibit several common attributes (Feller and Fitzgerald, 2001): (1) parallel development; (2) truly independent peer review; (3) prompt feedback to user and developer contributions; (4) highly talented developers; (5) user involvement; and, (6) rapid release schedules. In most literature group size is also seen as an important factor for innovation, rapid release and success (e.g., Linus' Law), but Krishnamurthy's (2002) research is raising a question on whether group size is a critical success factor. It may be that the active support of volunteers motivated largely because of intellectual stimulation and the desire to learn through independent study of source code and the process of a "truly independent peer review" (Feller and Fitzgerald, 2001) are more critical elements for success.

Table 2: Key Attributes and Components of OS Projects.

General Attributes

- Parallel development
- Independent peer review
- Prompt feedback
- Talented developers
- User involvement
- Rapid release schedules
- Large group size?
    - Participation largely voluntary but in some instances firms are paying employees to contribute
    - Intellectual stimulation and learning is a particularly strong motivator for participation, especially for volunteers
    - Utility another motivation for firms to pay employees to contribute

Lifecycle Attributes

Stage 1: Initiation

- Individual initiator or small group (e.g., 2-4 people)
- Highly skilled development team
- Modular design
- The kernel with some promise
- Virtual team may grow in size as progress is made (e.g., "trusted lieutenants")

Stage 2: Going "Open"

- Open source license
- Credibility of the initial kernel and participant base
- Communication systems
- Version control systems
- Initial governance structures and institutional design established

    - Operational rules
    - Collective-choice rules
    - Constitutional-choice rules

Stage 3: Project Growth, Stability or Decline

- Recruitment strategies
- Size of community with a "personal itch" for the product
- Project growth:
    - "OS network" — regional coordinators, communities of users expands
    - Real success — expands to "OS Enterprise"
        - Large network of developers/users
        - Core staff
        - Magazines, sponsors, organizational infrastructure
        - E.g., Linux, Apache, PhP
- Project stability:
    - Participation remains stable over time
- Project decline/death:
    - Loss of participant interest

Stage 1 of the OS lifecycle requires first one or several motivated individuals who often are friends or have already built social capital and have an idea for a project that fills in a need many have. Other "virtual members" might join in, and the project would evolve into a high quality development team, putting significant attention into product design (e.g., modularity) and the development of an initial project kernel that shows some promise. Stage 2 requires closer attention to the details of the OS license to be used, team communication, version control systems and approaches for participant recruitment. The designs of the project governance structure and rules outlining day-to-day operations, and collective-choice and constitutional-choice mechanisms could be very important factors but have not been researched. Stage 3 involves either community growth or stability, based on how well the project appeals to the personal interests of current and future members, and how well the project can maintain an enthusiastic community of participants through its institutional design and governance structure. Growth in this stage moves first toward a "network" where advanced coordination systems are required. "Trusted lieutenants" and "regional coordinators" are required to manage development and to assist with the growing user community. Finally, the truly successful projects (e.g., Linux, Apache, PhP) have evolved into "OS enterprise" endeavors where a core staff is established, an official logo is produced, in some cases project press or literature are generated (there are magazines now devoted to Linux), indicating that a massive end-user base has been achieved.

Is Open Source Really a Different Approach For Solving Complex Problems?

The argument has been made that the OS approach described above is simply a special case of the process of scientific or academic research (Bezroukov, 1999a; 1999b). For instance, perhaps the earliest example of an open source approach is the development of the dictionary (Economist, 1999). The standard process of scientific endeavor is to conduct research on your own or with a team of colleagues, submit findings in a peer-reviewed process, and make "open" the findings through published outlets. With the information and ideas in the public domain, others can build upon this research. How does OS differ from this current process? In our view, OS differs from the traditional scientific process in at least four important ways:

1. *OS collaboration over the Internet provides the ability to share the entire research product and the process of generating the product.* This differs from the traditional research process undertaken today in that everything about the development process could reside in the public domain. In traditional publishing outlets, there are substantial space limitations and traditionally only final results are presented to the community. Having the entire process open in the OS context promotes even more learning, especially in more junior participants who want to learn from more knowledgeable team members and provides even a more critical peer review of the entire process — not just

final results [10].
2. *OS collaboration over the Internet greatly increases the speed in which innovations can be published.* We do not need to wait for the traditional time period for paper-based products to be printed and distributed. And in certain circumstances, the peer-review process can be almost immediate.
3. *Open source collaboration over the Internet greatly increases the size of the potential audience the material might reach.* Researchers across the globe are gaining access to the Internet. They may not have access to paper versions of research publications.
4. *Open source licensing over the Internet allows for free sharing and distribution of intellectual property* (e.g., copying is allowed — even encouraged), *and requires that improvements to this research be given back to the community.* Traditional publication approaches do not have these kinds of requirements. Moreover, the OS licensing approach allows anyone who is interested in contributing to join in the ongoing project. Our current approach to research tends to be more proprietary, where people who are collaborating are working in "tighter," less open, groups.

In short, the OS approach is an extension of the traditional approach to research, but the way the entire collaboration process is conducted over the Internet coupled with the licensing requirements greatly enhances this process. This leads Lessig (2001) to observe that OS can be a fast and powerful process that promotes and encourages deep learning, thereby lowering barriers of entry for other innovators.

However researchers like Eunice (1998) and Lancashire (2001) have warned about the exaggerations of the promise of OS projects in past literature. A primary concern is that much of this earlier work on OS was based on a small number of high profile OS enterprise cases and, therefore, it is impossible to make robust arguments about the factors that led to their phenomenal success. Lancashire (2001), warns that "even trivial differences in the general license under which code is released affect subsequent development decisions... " and these decisions might affect the ultimate success or failure of the project. Further empirical research is needed to understand the social circumstances surrounding the technical and virtual organizational contexts that comprise OS development processes.

Empirical Research Strategy and Preliminary Results

Given the emergence of successful OS "enterprise" projects and the need for detail on the technical and social aspects of OS, we now introduce some important research questions to be pursued:

1. What are the critical success factors that enable some OS projects to generate substantially higher rates of participation, enthusiasm and "effective power" than most other OS projects?
2. Can the attributes of OS projects be borrowed and applied to other areas outside of computer programming, especially in areas where humanity is desperately trying to find solutions to complex problems? In other words, can the principles and approaches to collective action in OS projects be borrowed and applied to problems related to public policy and management?

We recently initiated a research program studying OS projects to address these questions. This study has two major objectives. The first is to understand the factors that lead to very successful endeavors like Linux or Apache and what factors result in project failures. Our hypothesis is that the institutional designs (operational, collective choice and constitutional choice rules) of these projects make a difference in the outcome. The second objective is to begin to study the diffusion of the OS approach into realms beyond computer science (non-programming cases) to understand how OS principles are being applied in non-computer programming cases — what we generally call open licensing projects. Such cases (experiments, really) are now beginning to emerge (Table 3). As they do, it is important to compare and contrast their experiences to the experiences in the OS programming realm to understand how these collaborative approaches to problem solving are similar and different.

Case Selection and Approach

Open licensing projects can be classified by three dimensions: (1) programming or non-programming; (2) stage of development (initiation, going open, and growth/decline); and, (3) outcome at that stage (success or failure). These three dimensions help to define a case selection matrix with the dependent variable being success (project and product growth or stability) or failure (project decline or death) at each stage [11] (Figure 1). Ultimately, our research project will study cases falling within each of the cells in Figure 1. Detailed case study is required to understand the factors that lead to successful and unsuccessful projects at various stages of the OS lifecycle. At the same time, we must sample enough cases that we will be able to say something definitive about what leads to the success or failure of these cases. This is especially important given that so much of the OS literature currently is based upon a small number of the successful "OS enterprise" cases like Linux (Eunice, 1998).

Table 3: OS-like Non-Programming Cases
Open Content Projects.

Non-Programming OS Cases		
Name	**URL**	**Type of Project**
Nupedia	http://www.nupedia.com	Online peer-reviewed encyclopedia
Wikipedia	http://www.wikipedia.com	Online encyclopedia
Open Directory Project	http://dmoz.org	Web link database
Open Content	http://www.opencontent.org	Open source licensing scheme for information content
OpenCourseWare	http://web.mit.edu/ocw	Open sharing of undergraduate and graduate course content
Opencode	http://eon.law.harvard.edu/opencode	Consortium for open research and content
Openlaw	http://eon.law.harvard.edu/openlaw	Experiment in the open crafting of legal arguments
Jake	http://jake-db.org/	Research software and database where content is built in an open source way
Open Music Registry	http://www.openmusicregistry.org	Open sharing of music using an Open Audio License
Open Content for Education	http://www.life-open-content.org/	
Public Library of Science	http://www.publiclibraryofscience.org/	Online archives of peer-reviewed scientific articles
Project Gutenberg	http://promo.net/pg/	Freely available digital content (started 1971)
World Lecture Hall	http://www.utexas.edu/world/lecture/	Online course materials
Linux documentation project	http://www.tldp.org/	Content project to develop documentation for Linux

Because OS programming cases are well established and further in the life cycle than the non-programming cases, we decided to concentrate the first phase of our study on the shaded cells in Figure 1. Sourceforge.net provides a good repository for hundreds of OS programming projects at various stages of the OS development lifecycle. For initial pretest of the case coding form, we decided to initially sample 15 cases based on the following criteria:

1. Cases should reflect the broad range of OSS development by software type (operating systems, databases, internet applications, desktop environments).
2. Cases should be relatively well known and therefore will be further along in the OS lifecycle and will have a relatively large participant base.
3. Cases should have a reasonable amount of information posted about them so we can undertake initial case coding without having to contact participants directly.
4. Cases should differ from each other by community size, geographic origin, ideology, product use, distribution, and management because that would give us more comprehensive picture of OSS.

Figure 1: Case Selection

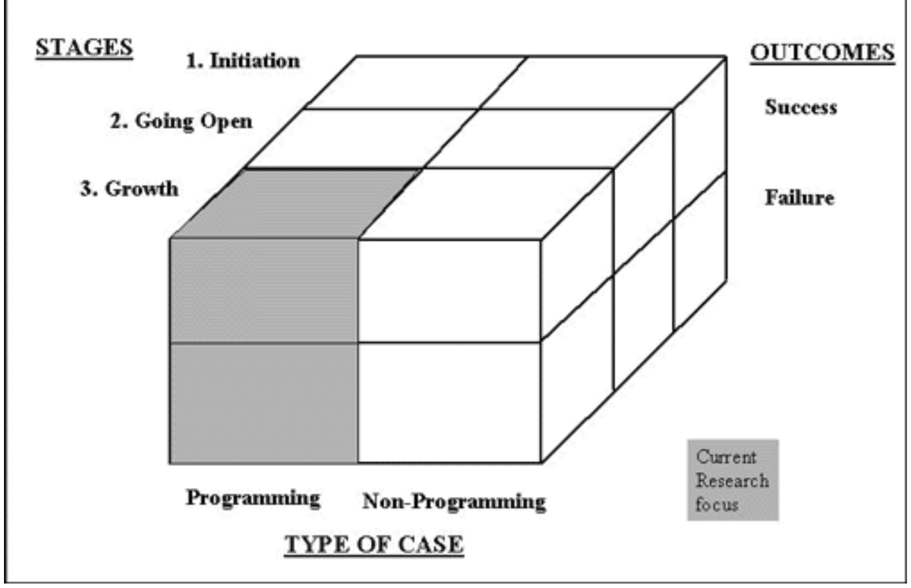

Table 4 lists the fifteen selected cases for initial coding. Most of these cases fall under the Programming/Success cell in Figure 1. At the time, we have analyzed these cases based only on our observations from activity within their project Web sites and from Web-based literature on these projects; we have not yet surveyed project participants.

Table 4: Open Source Programming Cases

Programming OS Cases		
Name	**URL**	**Type of Project**
Apache	http://www.apache.org	Web server
CVS	http://www.cvshome.org	OS programming management tool
Free BSD	http://www.freebsd.org	Unix operating system
GIMP	http://www.gimp.org	OS image manipulation software
GNOME	http://www.gnome.org	Unix desktop environment
GNU	http://www.gnu.org	Unix operating system
Greenstone	http://www.greenstone.org	Digital library software
KDE	http://www.kde.org	Unix desktop environment
Linux	http://www.linux.org	Unix operating system
Mozilla	http://www.mozilla.org	Web browser
MySQL	http://www.mysql.org	Database

Project Gutenberg	http://promo.net/pg/	Freely available digital content (started 1971)
Open Office	http://www.openoffice.org	Office application suite
PHP	http://www.php.net	OS programming tool
Python	http://www.python.org	OS programming tool
Zope	http://www.zope.org	OS application server

Preliminary Findings

For analytic consistency, we developed a case "coding form" to help guide case interpretation. This coding form collects detailed information on the attributes of the OS product and the attributes of the OS development process (including the institutional structure). To avoid inconsistency in case interpretation, all case analysis was undertaken by a single researcher with an understanding of computer programming. Here we will provide a brief summary of what we have found so far, organized by the OS lifecycle attributes in Table 2.

Stage 1 — Project Initiation:

Fourteen of the fifteen cases we studied appear to be initiated by either a single person or small group of people. The only exception is the Mozilla project (a Web browser), which was initiated by the company Netscape. For the most part, project initiators do appear to be technically sophisticated individuals who are motivated in part because of need but also because it is intellectually stimulating.

To our surprise, the modularity of a case was more difficult to measure than we expected. We thought we could simply find, or count, the number of modules in existing project repositories. Most of the cases utilize sourceforge.net as the system for project coordination and Sourceforge is designed around projects, not modules. Within a specific project, Sourceforge allows one to view discussions and downloads of various modules and these could potentially be counted. What is difficult is that there are different versions and different sets of libraries; without deeper knowledge about a specific project it is not easy to get an accurate count of modules with real confidence. The only case where we could get an accurate count was the Mozilla project where each module has a designated owner listed (Mozilla.org, 2002).

Stage 2 — Going Open:

Ten of the fifteen projects were initiated in the United States and then extended internationally. Five of these projects were designated "open" from the start. The others became open after they had been well defined. Twelve of these cases use the GNU General Public License (GPL) or a GPL compatible license (see Opensource.org, 2002d). PHP, Apache, and Zope each have established OS Definition compatible licenses, but have slight variations in their licensing requirements.

Communication between developers of these OS projects is handled through e-mail mailing lists and Web-based discussion forums. Face-to-face meeting is not common, with the exception of conferences related to OS software development. Users also discuss issues through mailing lists. In some OS cases, complicated sets of mailing lists exist for a variety of subjects related to the project (see, for example, http://mail.gnome.org/mailman/listinfo).

Consistent with what is reported by Feller and Fitzgerald (2001) we found that most OS projects use the Concurrent Version System software (CVS, 2002) to manage product content. The frequency of version releases for cases varies from one month (Linux) to one year (PHP, KDE), which does differ from the typical release schedule of proprietary software.

Understanding the governance structure and institutional designs of cases from publicly available online material is difficult. This could be because governance structures are not formally documented, or because they are documented in locations not in the public domain. But in each case we reviewed there do appear to be at least some social norms about various roles or participants. At each stage there has to be some common understanding of obligations and duties of various participants. In our cases, there are one or more designated project leaders and these tend to be people who initiated the project. These leaders tend to have the final word over issues on implementing project changes (suggested changes in code, operational rules, etc.) or when disputes arise (Maclachlan, 1999; Dafermos, 2001). In the case of OpenOffice.org, leaders of various subprojects are given voting rights over the future direction of the project and other issues (OpenOffice.org, 2002) but what voting rules are employed is not articulated in this document. This does suggest that rules do indeed exist related to collective-choice and constitutional-choice components in these projects and that these rules need to be studied in more depth. These are important components of teamwork and social capital generation that are vital to growing a development and user team.

Some projects we reviewed also have a "governing body," especially the well-established or mature OS projects such as Linux and Apache. For example, the Apache HTTP server project was first initiated in 1995. Then, in 1999, members of the Apache Group formed the Apache Software Foundation (ASF) and established a board of directors to provide organizational, legal, and financial support for the Apache HTTP Server, their core project. While the Board retains ultimate responsibility for the foundation, it delegates decision-making authority for the technical direction of projects to Project Management Committees (PMCs). The PMCs are groups of members who take responsibility for the long-term direction and management of a specific OS project. While the Apache Server remains the core product, the ASF is now involved in other OS projects (e.g., Jakarta, XML, etc.) run by different PMCs but all under the control of the Board of Directors.

Some project leaders coordinate activity in OS projects and check submissions of other developers. In other cases, this is delegated to some of the other developers themselves. In Linux, for example, members of such groups are called "trusted lieutenants." This is a layer of project developers who are the interface between project leaders and other developers and/or users. In most cases trusted lieutenants appear to be formally appointed by project leaders or by a governing board but more research is needed to understand these procedures better. They are also recognized as highly competent and trustworthy by their peers.

In most cases, the designated project leader appears to have authority to define the operational rules. For instance, the decision about the release of a new version tends to fall under the leader's domain. It also appears that project leaders are the authority at the collective-choice level: they can establish and change operational-level rules. Linux was the only project where we could identify a formal written Constitution online, and this appears to be user-initiated and governs only particular subgroups of the project and not the entire project (Linux Australia, 2002; UMBC, 1997). This does not mean necessarily that in these projects there are no established constitutional-choice or collective-choice rules. It means that these rules are either: (1) formally written but not put online, or (2) they exist as "social norms" and are not written down.

Stage 3 — Growth, Stability or Decline:

All of the projects we investigated invite new users or developer participants to join in and do not specify any requirements of participants (e.g., advanced technical skills). To become a user of Linux or Apache, for example, requires only the basic computer skills needed to download and install software. Consequently, in most cases, it is difficult to determine the number of users because counters of downloads are usually unavailable and other people can copy and share software once it is downloaded. Estimating the number of developers is equally difficult — it could be done, but it would require reviewing many discussion lists and would be extremely time-intensive. In OS enterprise cases there are sometimes projections on users/participants. For example, in Linux, the approximate number of current users is projected to be around eighteen million (Linux Counter, 2002).

The contributors to source code in these fifteen cases are distributed geographically all over the world; however, only four projects have a substantial contribution from non-U.S. and non-European developers. Countries that have participants in the cases we reviewed include: U.S., Canada, Australia, India, China, Japan, Russia, and South Korea. This finding is consistent with the geographic patterns described by Lancashire (2001) regarding the Linux and Gnome projects.

Most of these contributors are volunteers, stimulated by intellectual interest. But often these contributors benefit from their efforts in related businesses, for example, publishing articles about OS software development. But our analysis of some groups supports findings by Lakhani et al. (2002) that there is some participation by people employed by businesses. This is an interesting phenomenon in that some businesses are willing to contribute employee work time toward OS projects because they are as a company utilizing an OS product and have a need for new functionality. Our research also suggests that many OS developers know each other from past computer science work or through interaction in higher education. Project participants in our cases averaged three hours per day to the project according to available literature on cases.

As the project teams move from team to network to OS enterprise, complications arise in terms of language. With the exception of the Greenstone project, all of our cases have localized mailing lists using different languages. In the extreme example of Linux, there are lists supporting Albanian, Catalan, Chinese, Chinese, Croatian, Czech, Danish, Dutch, English, French, German, Greek, Hungarian, Indonesian, Italian, Japanese, Korean, Norwegian, Polish, Portuguese, Romanian, Russian, Spanish, Swedish, Turkish, and Vietnamese languages. This requires a new level of project coordination and complexity, where there are designated team members who act as moderators or translators as modules are developed and tested and as new documentation is created.

Finally, to our surprise, all of the fifteen projects have some permanent staff to manage the project. This is probably the result of our decision to look only at more established OS projects. The average number of such people in the cases we reviewed is somewhere between five and nine. For example, OpenOffice.org core staff includes five people who are "community managers," and "coordinators and release managers." Community managers are responsible for the day-to-day management, growth and planning of the project. Coordinators and release managers are responsible for coordinating developers and ensuring a smooth release schedule (OpenOffice.org, 2002). The Mozilla project has fourteen people as a permanent staff (Mozilla.org, 2002). For most cases it appears that these staff are paid either from donation sources or through support from some parallel business.

Next Steps

The findings described above are preliminary and based upon available literature on the projects over the Internet. Over the coming year we will be continuing this research agenda. Specific tasks will include:

- Refinement of the case coding form (removing questions that do not seem applicable) based on the lessons from these first 15 cases.
- Development of an online survey form that complements the coding form. Of central interest will be questions that cannot be easily determined through online research. The institutional design components of OS projects do not appear to be well documented (e.g., governance structures) will be a central focus in this stage of the study.
- Increasing sampling of programming cases and undertaking a random sampling approach to OS cases.
- Beginning similar analysis of non-programming cases that take an "OS-like" approach. The population of possible cases will be much smaller than in the realm of programming and many of these projects will be in the early stages of an "OS-like" lifecycle. Consequently we will analyze as many non-programming projects as can be identified and can be accomplished under current project resources.

The preliminary findings from our review of the fifteen programming cases raise some issues requiring further research either through coding form refinement or through the online survey instrument.

Existing literature on OS has emphasized the importance of modularity, but in this initial research we had difficulties measuring it. For example, it does not appear that Sourceforge provides summary data on the current number of modules contained within a particular project. However, the methods for modular designs may be critical to a successful project because of the importance of parallel development in OS projects. Efforts to quantify and explain modularity will be important as we further evaluate cases.

The timing of when projects go open (the move to Stage 2 in Table 2) appears to vary and it is not clear if this timing is an important factor leading to long-term success or failure. Five of these mature projects we studied became open immediately, so the community was encouraged to participate in project design. As we get a better handle on measures of success, it will be interesting to see whether these "open early" projects do better, worse, or the same as projects who wait to go open until the project kernel is better designed. A related issue is how to measure "project credibility" at the time of going open.

Information on the operational, collective-choice and constitutional-choice rules of projects are difficult to find in the publicly available Internet forums of various cases. We expect that these rules do exist either formally or informally, but to understand them will require the next phase of investigation — surveys of project participants.

Measuring growth, especially in the user community is difficult, for in most cases the software is free to download, and there is no registration requirement. There may be a similar problem of tracking use as the OS concepts get transferred into non-programming domains.

Finally, as a project grows in participant base internationally, there will be the need for language coordinators and translators to help manage parallel development of a virtual team speaking (and writing) in multiple languages. In the context of programming cases, this might be easier if a programming language serves as a common language between participants. But even so, coordination and communication across linguistic boundaries may be substantial and success or failure of a project to gain a critical mass of participants may hinge on the abilities of these coordinators.

Conclusion

The concept of "Open Licensing" coupled with the global communication power of the Internet brings new opportunities for collaboration and complex problem solving in the context of computer programming. The existence of enterprise OS cases like Linux and Apache Web Server suggest that there is something to this idea. The question is whether these cases are flukes, or whether open licensing is a new emerging paradigm for collaborative Internet-based efforts to solve complex problems. We would argue that these cases are not flukes, but rather the conditions were right for them to acquire large amounts of (mostly) donated intellectual labor and that they had in place systems of management and coordination which were configured appropriately to meet the needs of the project. But currently a majority of the literature on OS reviews high-profile, mature, success-story projects that provide a possibly skewed perception of how these projects operate.

To help understand this more fully, we provide an overview of what has been published on the institutional designs (the operations and management) of OS programming projects. From this, we summarize key attributes and components in the life cycle of OS projects. With these defined, using information we could find on the Internet, we analyze fifteen programming cases to see if they support or refute what is already known about how OS projects operate. We report preliminary findings based on the stages of the open source life cycle.

The main motivation behind this paper is to learn about how high-profile, "enterprise" OS programming cases such as Linux or Apache Web Server achieved their success in collective action. This is important because the principles of Open Licensing have utility beyond programming — they are potentially useful in solving other complex problems that humanity faces, particularly in the context of public policy and management. In this paper we note a number of non-programming OS-like projects that have recently emerged, which are experiments in this direction. It is probably too early to try and measure the success of the non-programming projects, but it will hinge on how well they are able to motivate others to participate and how well they manage the various stages of the OS lifecycle. None of these innovative projects we have identified appear, however, to address problems of public policy or management. So let us conclude this paper by describing a vision of the future — how an Open License approach, crafted and managed appropriately, might enhance our ability to harness a global collaborative effort to solve complex environmental management and policy problems.

An Example of a Future Vision: The Idea of "Open Content Environmental Management"

Over the last decade, we have been involved in a research program called the International Forestry Resources and Institutions (IFRI) program (http://www.indiana.edu/~ifri/). This program studies how human communities have been able to craft institutional designs for the effective management of forest resources. "Institutions," in this context are defined generally as "rules in use," with mechanisms also in place for the monitoring and enforcement of these rules. Environmental institutions can be laws created by national or state governments, they can be rules established around property-rights and market-based structures, or they can simply be well-understood "social norms" followed and enforced by people who live together in a community (Ostrom, 1998, 1992, 1990; Schweik et al., 1997). In developing country contexts, communal property and locally established mechanisms for monitoring and enforcing of established rules may be prominent (Gibson, McKean and Ostrom, 2000). In the United States, property rights, markets and national and state laws are the dominant institutional forms. But even under these broad institutional requirements, there are opportunities for communities to craft more detailed institutional designs for environmental management.

Habitat conservation planning (HCP), a requirement of the U.S. Endangered Species Act, provides a good example. HCPs are institutional designs crafted by conservationists, developers and public officials, with the goal of balancing economic development and endangered species protection (Thomas, forthcoming). Over 250 HCPs now exist in the United States, and their institutional designs are well documented (Schweik and Thomas, 2002). But currently, there is limited sharing and learning from various cases, other than what is produced in case studies or comparative literature. There may be a great many lessons learned and much knowledge to be gained if there was a mechanism to share, in an open-source fashion, these kinds of institutional designs.

Imagine applying an "Open Content" approach to this endeavor. It would involve an "Open Publication License" (Grossman, 1998; Economist, 1999; Cisneros, 1999), which follows generally the same principles as the Open Source Definition described earlier, but applies them to information content (e.g., documents, databases, etc.) rather than computer programs (Newmarch, 2000).

In the initiation stage of the project, an initial database with plausible promise would need to be built documenting various approaches to environmental management and corresponding institutional designs. At the same time, before embarking on the project there would need to be some serious considerations about the motivations of future collaborators to join in on such a project. For example, to solicit academic participation, the design of the collaborative platform would probably have to take the design of a new form of electronic peer-reviewed journal, to allow for researchers to claim publications on their curriculum vitae (Schweik and Grove, 2000). Some serious design work would be required to develop a content database that would be organized in a modular way. For example, a database could be constructed that provided information about the design of most existing Habitat Conservation Plans. These tend to be well documented and could be stored in a database in a standardized fashion. People with expertise in this area of environmental management would need to be involved in this stage to help make good modular design decisions and to make the project credible.

The "Going Open" stage of the project would involve consideration of the appropriate "Open-Publication" license. OPL (Open Content, 2002b) and the GNU Free Documentation License (Free Software Foundation, 2002) are two such licenses that provide a starting point. Newmarch (2001, 2000) provides a helpful overview of some of these licenses and discussions about the open content issue. Critical to the particular license will be the rules for providing credit to people who contribute valuable intellectual property (e.g., an academic researcher who needs publications for tenure). Communication systems and content management systems would need to be created, and the project governance and institutional structure would need to be defined.

The growth stage of the project would involve similar considerations about such matters as motivations, recruitment, and motivations as these apply to OS projects. Once open, new institutional designs (e.g., the documentation for a newly formed HCP), evaluations of existing institutional designs, policy or management documents, field data, case studies or relevant theoretical work could be posted for peer-review (e.g., a new Habitat Conservation Plan). The lower cost of publishing over the Internet provides an opportunity to create a new form of collaboration that is focused around the Open Publication concept. Other users of such a system would be guaranteed the right to freely download, utilize or share online content under the rules of the Open Publication License in place. Content could also be modified with the licensing stipulation that such modifications be then made public. For example, a community in the eastern US could review the institutional designs of several posted HCPs from the west, download their content, choose one that they think is appropriate for their own context, make modifications to fit their own situation and goals, and then post their modified designs back to the system for other collaborators to see, evaluate and possibly use in the future.

This may be the direction where some new Open Content initiatives listed earlier in Table 3 may be headed. But these ideas are so new that little experience exists outside of OS programming endeavors. In order for these ideas to reach "enterprise" scales in other complex problem-solving contexts, more understanding is needed about the detailed processes and factors that lead to successful OS programming projects. This paper, and the research program it describes, is trying to help identify the critical factors that need to be in place to achieve successful open collaboration in any contexts (programming or content). Just imagine the progress we might make if we can establish "enterprise" open source-like content projects working collaboratively to address serious and complex public policy and management problems facing humanity.

About the Authors

Charles M. Schweik is an Assistant Professor in the Department of Natural Resources Conservation and the Center for Public Policy and Administration at the University of Massachusetts, Amherst. He holds a PhD in Public Policy from Indiana University, a Masters of Public Administration from Syracuse University and a BA in Computer Science. Prior to academia he was a programmer and project leader at IBM.
E-mail: cschweik@pubpol.umass.edu
Direct comments to cschweik@pubpol.umass.edu

Andrei Semenov is a graduate student at the University of Massachusetts, Amherst in the Department of Landscape Architecture and Regional Planning and has extensive experience in programming Web applications.

Acknowledgements

This material is based upon work supported by the Cooperative State Research Extension, Education Service, U.S. Department of Agriculture, Massachusetts Agricultural Experiment Station, under Project No. MAS00847. Additional support comes from the Burlington Laboratory (4454) and the Global Change Program, Northeastern Forest Research Station, USDA Forest Service.

We thank Alexander Stepanov and Theodora Campbell-Orde for their efforts related to this project, and Robert and Joanne Schweik for comments from earlier drafts. We are also grateful to the Department of Natural Resources Conservation and the Center for Public Policy and Administration at the University of Massachusetts, Amherst for research support, and in particular, Bill McComb, for ongoing encouragement of this project.

Notes

1. The number of existing OS software projects is sizable. The prominent OS software Web site www.sourceforge.net stated recently (4 October 2002) that there are more than 48,000 OS projects utilizing their services.

2. The concepts of free software and its relation (and differences) to OS are complicated and not important to this discussion. Readers are encouraged to read Perens (1999) or Pavlicek (2000) for more information on these distinctions.

3. Feller and Fitzgerald, 2001, p. 17.

4. This section is based loosely on the discussion on the OS development lifecycle provided by Feller and Fitzgerald (2001).

5. Torvalds, 1999, p. 108.

6. Feller and Fitzgerald, 2001, pp. 77-78.

7. Ostrom et al., 1994, p. 38.

8. Feller and Fitzgerald, 2001, p. 91.

9. See http://www.firstmonday.org/issues/issue7_9/letters/.

10. This is exactly why the World Wide Web became so successful. Specifically designed in the Internet Explorer and Netscape browsers are a "view source" option, which allows any web user the ability to see the programming logic (HTML) for a Web page they are visiting. In the early days of Web publishing, this viewing of source allowed others to learn directly from the Web page code of others. This feature greatly enhanced the speed in which Web pages were published around the world (Lessig, 2001).

11. Several criteria can be used as a measure of project success, including: (1) annual growth of participant base; (2) annual growth of user community; (3) growth in "market share" (in the case of programming cases); (4) user satisfaction with the product; and (5) peer recognition of the product. Not all of these are prefect indicators, however. For example, a

participation base growth may stabilize, but ultimately, the project could still produce a quality product. Of course, we do not want to sample on the dependent variable, so these criteria were not used to select specific cases.

References

B. Behlendorf, 1999. "Open source as a business strategy," In: M. Stone, S. Ockman, and C. DiBona (editors). *Open Sources: Voices from the Open Source Revolution.* Sebastopol, Calif.: O'Reilly & Associates, and at http://www.oreilly.com/catalog/opensources/book/brian.html, accessed 28 November 2002.

N. Bezroukov, 1999a. "Open Source Software as a Special Type of Academic Research (a Critique of Vulgar Raymondism)," *First Monday,* volume 4, number 10 (October), at http://firstmonday.org/issues/issue4_10/bezroukov/, accessed 8 December 2002.

N. Bezroukov, 1999b. "A Second Look at the Cathedral and the Bazaar," *First Monday,* volume 4, issue 12 (December), at http://firstmonday.org/issues/issue4_12/bezroukov/, accessed 8 December 2002.

D. Bollier, D. 1999. "The Power of Openness: Why Citizens, Education, Government and Business Should Care About the Coming Revolution in Open Source Code Software," http://eon.law.harvard.edu/opencode/h3o/, accessed 9 September 2002.

C.B. Browne, 1998. "Linux and Decentralized Development," *First Monday,* volume 3, number 3 (March), at http://firstmonday.org/issues/issue3_3/browne/, accessed 9 September 2002.

F.J. Cavalier, 1998. "Some Implications of Bazaar Size," at http://www.mibsoftware.com/bazdev/ accessed 28 December 2002.

O.S. Cisneros, 1999. "Expanding the Universe of Ideas," at http://www.wired.com/news/politics/ 0,1283,20276,00.html, accessed 25 September 2002.

CVS, 2002. "Concurrent Versions System," at http://www.cvshome.org/, accessed 25 September 2002.

G.N. Dafermos, 2001. "Management and Virtual Decentralized Networks: The Linux Project," Masters Thesis, Durham Business School, http://opensource.mit.edu/papers/dafermoslinux.pdf, accessed 23 September 2002.

Economist, 1999. "Hacker Journalism," *Economist,* at http://www.economist.com/displayStory.cfm?Story_ID=265022, accessed 25 September 2002.

Economist, 1998. "Red Hat Trick: Linux Operating System May Pose a Serious Threat to Microsoft Dominance," *Economist,* volume 348, number 8088 (3 October), p. 76.

K. Edwards, 2001. "Epistemic Communities, Situated Learning and Open Source Software," at http://opensource.mit.edu/papers/kasperedwards-ec.pdf, accessed 9 September 2002.

J. Eunice, 1998. "Beyond the Cathedral, Beyond the Bazaar," at http://www.illiminata.com/, accessed 25 August 2002.

J. Feller and B. Fitzgerald, 2001. *Understanding Open Source Software Development.* London: Addison-Wesley.

K. Fogel, 1999. *Open Source Development with CVS.* Scottsdale, Ariz.: Coriolis Group.

Free Software Foundation, 2002. "GNU Free Documentation License," at http://www.gnu.org/copyleft/fdl.html, accessed 27 September 2002.

C.C. Gibson, M.A. McKean, and E. Ostrom (editors), 2000. *People and Forests: Communities, Institutions, and Governance.* Cambridge, Mass.: MIT Press.

M.W. Godfrey and Q. Tu, 2000. "Evolution in Open Source Software: A Case Study," at http://plg.uwaterloo.ca/~migod/papers/icsm00.pdf, accessed 25 August 2002.

L. Grossman, 1998. "New Free License to Cover Content Online," at http://www.onmagazine.com/on-mag/reviews/article/0,9985,621,00.html, accessed 25 September 2002.

L.L. Kiser and E. Ostrom, 1982. "The Three Worlds of Action: A Meta-theoretical Synthesis of Institutional Approaches.," In: E. Ostrom (editor). *Strategies of Political Inquiry.* Beverley Hills, Calif.: Sage, pp. 179-222.

S. Krishnamurthy, 2002. "Cave or Community? An Empirical Examination of 100 Mature Open Source Projects," *First Monday,* volume 7, number 6 (June), at http://firstmonday.org/issues/issue7_6/krishnamurthy/, accessed 9 September 2002.

K.R. Lakhani and E. von Hippel. 2002. "How Open Source Software Works: Free User-to-User Assistance," at http://opensource.mit.edu/papers/lakhanivonhippelusersupport.pdf, accessed 2 October 2002.

K.R. Lakhani, B. Wolf, J. Bates, and C. DiBona, 2002. "The Boston Consulting Group Hacker Survey, Release 0.73," at http://www.osdn.com/bcg/bcg-0.73/BCGHackerSurveyv0-73.html, accessed 9 September 2002.

D. Lancashire, 2001. "Code, Culture and Cash: The Fading Altruism of Open Source Development," *First Monday,* volume 6, number 12 (December), at http://firstmonday.org/issues/issue6_12/lancashire/, accessed 25 September 2002.

M. Learmonth, 1997. "Giving It All Away," at http://www.metroactive.com/papers/metro/05.08.97/cover/linus-9719.html, accessed 18 August 2002.

L. Lessig, 2001. *The Future of Ideas: The Fate of the Commons in a Connected World.* New York: Random House.

Linux Australia. 2002. "Linux Australia Constitution," at http://www.linux.org.au/org/constitution.phtml, accessed 29 September 2002.

Linux Counter. 2002. "The Linux Counter," at http://counter.li.org/estimates.php, accessed 29 September 2002.

M. Maclachlan, 1999. "Panelists Describe Open Source Dictatorships," at http://www.techweb.com/wire/story/TWB19990812S0003, accessed 6 September 2002.

A. Mockus, R.T. Fielding, and J. Herbsleb. 2000. "A Case Study of Open Source Software Development: The Apache Server," at http://opensource.mit.edu/papers/mockusapache.pdf, accessed 9 September 2002.

G. Moody, 2001. *Rebel Code: Linux and the Open Source Revolution.* Cambridge, Mass.: Perseus Press.

Mozilla.org, 2002. "Mozilla.org Staff Members," at http://www.mozilla.org/about/stafflist.html, accessed 02 October 2002.

J. Newmarch, 2001. "Lessons from Open Source: Intellectual Property and Courseware," First Monday, volume 6, number 6 (June), at http://firstmonday.org/issues/issue6_6/newmarch/, accessed 27 September 2002.

J. Newmarch, 2000. "Open Content Licenses," http://jan.netcomp.monash.edu.au/opendoc/paper.html, accessed 27 September 2002.

Open Content, 2002a. at www.opencontent.org, accessed 25 September 2002.

Open Content, 2002b."Open Content License (OPL)," at http://opencontent.org/opl.shtml, accessed 25 September 2002.

Open CourseWare, 2002. at http://web.mit.edu/ocw, accessed 25 September 2002.

Open Law, 2002. at http://eon.law.harvard.edu/openlaw, accessed 25 September 2002.

Open Music, 2002. at http://www.openmusicregistry.org, accessed 25 September 2002.

OpenOffice.org, 2002. at http://www.openoffice.org, accessed 2 October 2002.

Opensource.org, 2002a. http://www.opensource.org/, accessed 14 September 2002.

Opensource.org. 2002b. "Holloween Document I (Version 1.14)," at http://www.opensource.org/holloween1.php, accessed 25 August 2002.

Opensource.org, 2002c. "The Open Source Definition (Version 1.9)," at http://www.opensource.org/docs/definition.html, accessed 25 September 2002.

Opensource.org, 2002d. "The Approved Licenses," at http://www.opensource.org/licenses/index.html, accessed 25 September 2002.

T. O'Reilly and E. Dyson, 1998. "The Open-Source Revolution," *Release 1.0,* at http://www.edventure.com/release1/1198.html, accessed 14 March 2000.

E. Ostrom, 1998. "A Behavioral Approach to the Rational Choice Theory of Collective Action," *American Political Science Review,* volume 92, number 1, (March), pp. 1-22.

E. Ostrom, 1992. *Crafting Institutions for Self-Governing Irrigation Systems.* San Francisco: ICS Press.

E. Ostrom, 1990. *Governing the Commons: The Evolution of Institutions for Collective Action.* New York: Cambridge University Press.

E. Ostrom, R. Gardner, and J. Walker, 1994. *Rules, Games and Common Pool Resources.* Ann Arbor: University of Michigan Press.

R.C.Pavlicek, 2000. *Embracing Insanity: Open Source Software Development.* Indianapolis, Ind.: Sams.

B. Perens, 1999. "The Open Source Definition.," In: C. DiBona, S. Ockman and M. Stone (editors). *Open Sources: Voices from the Open Source Revolution.* Sebastopol, Calif.: O'Reilly & Associates.

E.S. Raymond, 1998a. "The Cathedral and the Bazaar," *First Monday,* volume 3, number 3 (March), at http://firstmonday.org/issues/issue3_3/raymond/, accessed 28 November 2002.

E.S. Raymond, 1998b. "Homesteading the Noosphere," *First Monday,* volume 3, number 10 (October), at http://firstmonday.org/issues/issue3_10/raymond/, accessed 9 Septmeber 2002.

H. Rheingold, 1993. *The Virtual Community: Homesteading on the Electronic Frontier.* New York: Harper Perennial.

Sourceforge.net, 2002. http://www.sourceforge.net, accessed 20 September 2002.

C. Schweik and C. Thomas, 2002. "Using Remote Sensing to Evaluate Environmental Institutional Designs: An Habitat Conservation Planning Example," *Social Science Quarterly,* volume 83, pp. 244-262.

C. Schweik and J.M. Grove, 2000. "Fostering Open-Source Research Via a World Wide Web System," *Public Administration and Management,* volume 5, number 3, at http://www.pamij.com/5_4/5_4_2_opensource.html, accessed 2 December 2002.

C. Schweik, K.R. Adhikari, and K.N. Pandit, 1997. "Land-cover Change and Forest Institutions: A Comparison of Two Sub-basins in the Siwalik Hills of Nepal," *Mountain Research and Development,* volume 17, number 2, pp. 99-116.

C. Thomas, forthcoming. *Bureaucratic Landscapes: Interagency Cooperation and the Preservation of Biodiversity.* Cambridge, Mass.: MIT Press.

L. Torvalds, 1999. "The Linux Edge," In: C. DiBona, S. Ockman and M. Stone (editors). *Open Sources: Voices from the Open Source Revolution.* Sebastopol, Calif.: O'Reilly & Associates.

UMBC, 1997. "UMBC Linux User Group Constitution," at http://linux.umbc.edu /constitution.html, accessed 29 September 2002.

Editorial history

Paper received 11 December 2002; accepted 24 December 2002.

The Institutional Design of Open Source Programming: Implications for Addressing Complex Public Policy and Management Problems by Charles M. Schweik and Andrei Semenov.
First Monday, Volume 8, Number 1 - 6 January 2003
http://ojphi.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/viewArticle/1019/2426