

UPGRADE is the European Journal for the Informatics Professional, published bimonthly at <http://www.upgrade-cepis.org/>

UPGRADE is the anchor point for UPENET (UPGRADE European Network), the network of CEPIS member societies' publications, that currently includes the following ones:

- **Mondo Digitale**, digital journal from the Italian CEPIS society AICA
- **Novática**, journal from the Spanish CEPIS society ATI
- **OCG Journal**, journal from the Austrian CEPIS society OCG
- **Pliroforiki**, journal from the Cyprus CEPIS society CCS
- **Pro Dialog**, journal from the Polish CEPIS society PTI-PIPS

Publisher

UPGRADE is published on behalf of CEPIS (Council of European Professional Informatics Societies, <http://www.cepis.org/>) by **Novática** (<http://www.ati.es/novatica/>), journal of the Spanish CEPIS society ATI (*Asociación de Técnicos de Informática*, <http://www.ati.es/>)

UPGRADE monographs are also published in Spanish (full version printed; summary, abstracts and some articles online) by **Novática**, and in Italian (summary, abstracts and some articles online) by the Italian CEPIS society ALSI (*Associazione nazionale Laureati in Scienze dell'informazione e Informatica*, <http://www.alsi.it/>) and the Italian IT portal **Tecnoteca** (<http://www.tecnoteca.it/>)

UPGRADE was created in October 2000 by CEPIS and was first published by **Novática** and **INFORMATIK/INFORMATIQUE**, bimonthly journal of SVI/FISI (Swiss Federation of Professional Informatics Societies, <http://www.svifis.ch/>)

Editorial Team

Chief Editor: Rafael Fernández Calvo, Spain, rfrcalvo@ati.es

Associate Editors:

François Louis Nicolet, Switzerland, nicolet@acm.org
 Roberto Carniel, Italy, rcarniel@dgt.uniud.it
 Zakaria Maamar, Arab Emirates, <Zakaria.Maamar@zu.ac.ae>
 Soraya Kouadri Mostéfaoui, Switzerland, soraya.kouadrimostefaoui@unifr.ch

Editorial Board

Prof. Wolfried Stucky, CEPIS Past President
 Prof. Nello Scarabottolo, CEPIS Vice President
 Fernando Piera Gómez and
 Rafael Fernández Calvo, ATI (Spain)
 François Louis Nicolet, SI (Switzerland)
 Roberto Carniel, ALSI – Tecnoteca (Italy)

UPENET Advisory Board

Franco Filippazzi (Mondo Digitale, Italy)
 Rafael Fernández Calvo (Novática, Spain)
 Veith Risak (OCG Journal, Austria)
 Panicos Masouras (Pliroforiki, Cyprus)
 Andrzej Marciniak (Pro Dialog, Poland)

English Editors: Mike Andersson, Richard Butchart, David Cash, Arthur Cook, Tracey Darch, Laura Davies, Nick Dunn, Rodney Fennemore, Hilary Green, Roger Harris, Michael Hird, Jim Holder, Alasdair MacLeod, Pat Moody, Adam David Moss, Phil Parkin, Brian Robson

Cover page designed by Antonio Crespo Foix, © ATI 2005

Layout Design: François Louis Nicolet

Composition: Jorge Llácer-Gil de Ramales

Editorial correspondence: Rafael Fernández Calvo rfrcalvo@ati.es

Advertising correspondence: novatica@ati.es

UPGRADE **Newsletter** available at

<http://www.upgrade-cepis.org/pages/editinfo.html#newsletter>

Copyright

© Novática 2005 (for the monograph and the cover page)

© CEPIS 2005 (for the sections MOSAIC and UPENET)

All rights reserved. Abstracting is permitted with credit to the source. For copying, reprint, or republication permission, contact the Editorial Team

The opinions expressed by the authors are their exclusive responsibility

ISSN 1684-5285

Monograph of next issue (August 2005):

"Normalisation & Standardisation in IT Security"

(The full schedule of UPGRADE is available at our website)

Monograph: *Libre Software as A Field of Study* (published jointly with *Novática**, in cooperation with the European project CALIBRE)

Guest Editors: *Jesús M. González-Barahona and Stefan Koch*

- 2 Presentation
Libre Software under The Microscope — *Jesús M. González-Barahona and Stefan Koch*
- 5 CALIBRE at The Crest of European Open Source Software Wave — *Andrea Deverell and Par Agerfalk*
- 6 *Libre Software Movement: The Next Evolution of The IT Production Organization?* — *Nicolas Jullien*
- 13 Measuring *Libre Software* Using Debian 3.1 (Sarge) as A Case Study: Preliminary Results — *Juan-José Amor-Iglesias, Jesús M. González-Barahona, Gregorio Robles-Martínez, and Israel Herráiz-Tabernero*
- 17 An Institutional Analysis Approach to Studying *Libre Software* 'Commons' — *Charles M. Schweik*
- 28 About Closed-door Free/*Libre*/Open Source (FLOSS) Projects: Lessons from the Mozilla Firefox Developer Recruitment Approach — *Sandeep Krishnamurthy*
- 33 Agility and *Libre Software* Development — *Alberto Sillitti and Giancarlo Succi*
- 38 The Challenges of Using Open Source Software as A Reuse Strategy — *Christian Neumann and Christoph Breidert*

MOSAIC

- 43 Computational Linguistics
Multilingual Approaches to Text Categorisation — *Juan-José García-Adeva, Rafael A. Calvo, and Diego López de Ipiña*
- 52 Software Engineering
A Two Parameter Software Reliability Growth Model with An Implicit Adjustment Factor for Better Software Failure Prediction — *S. Venkateswaran, K. Ekambavanan, and P. Vivekanandan*
- 59 News & Events: Proposal of Directive on Software Patents Rejected by The European Parliament

UPENET (UPGRADE European Network)

- 61 From **Pliroforiki** (CCS, Cyprus)
Informatics Law
Security, Surveillance and Monitoring of Electronic Communications at The Workplace — *Olga Georgiades-Van der Pol*
- 66 From **Mondo Digitale** (AICA, Italy)
Evolutionary Computation
Evolutionary Algorithms: Concepts and Applications — *Andrea G. B. Tettamanzi*

* This monograph will be also published in Spanish (full version printed; summary, abstracts, and some articles online) by **Novática**, journal of the Spanish CEPIS society ATI (*Asociación de Técnicos de Informática*) at <http://www.ati.es/novatica/>, and in Italian (online edition only, containing summary, abstracts, and some articles) by the Italian CEPIS society ALSI (*Associazione nazionale Laureati in Scienze dell'informazione e Informatica*) and the Italian IT portal Tecnoteca at <http://www.tecnoteca.it/>.

An Institutional Analysis Approach to Studying *Libre* Software ‘Commons’

Charles M. Schweik



This paper is copyrighted under the Creative Commons Attribution-NonCommercial-NoDerivs 2.5 license available at <<http://creativecommons.org/licenses/by-nc-nd/2.5/>>

Anyone interested in Libre software will be interested in the question of what leads to success and failure of Libre projects. This paper marks the beginning of a five-year research program, funded by the U.S. National Science Foundation, to identify design principles that lead to successful Libre software development efforts. Recently, scholars have noted that Libre software projects can be considered a form of ‘commons’, producing software public goods. This connection is important, for a large body of theoretical and empirical findings exists related to long-enduring environmental commons which could also apply to and inform Libre software projects. Institutions – defined here as rules-in-use – are a central set of variables known to influence the ultimate outcome of commons settings (e.g., long-enduring commons or ones that succumb to what G. Hardin has called the “Tragedy of the Commons”). To date, we know relatively little about the institutional designs of Libre projects and how they evolve. This paper presents an oft-used framework for analyzing the institutional designs of environmental commons settings that will guide upcoming empirical research on Libre software projects. It presents a trajectory of these projects and discusses ways to measure their success and failure. The paper closes by presenting example hypotheses to be tested related to institutional attributes of these projects.

Keywords: Common Property, Commons, Institutions, Libre Software.

1 Introduction

Several articles in *UPGRADE* and *Novática*’s 2001 issue on Open Source/Free Software [1] (referred to from now on as simply ‘Libre’ software) noted that the composition of development teams was changing, from all-volunteer teams to teams with paid participants from industry, government or not-for-profit organizations [2]. While the *Libre* collaborative approach is not a panacea, there are enough success stories to conclude that this development paradigm is viable and important. At the same time, a much higher number of *Libre* projects have been abandoned before reaching the goals they set out to achieve at their outset [28]. Therefore, an important question, recognized by a number of researchers [3-8] is: what factors lead to success or failure of *Libre* projects?

Recently *Libre* software development projects have been recognized as a form of ‘commons’, where sets of volunteer and paid professional team members from all over the globe collaborate to produce software that is a public good [9-13][53]. This recognition provides the opportunity to connect separate streams of research on the management of natural resource commons ([14][15] provide summaries) with the more traditional information system research related to the production of software, and *Libre* software in particular.

Viewing *Libre* projects as a commons focuses attention on attributes and issues related to collective action, governance, and the often complex and evolving system of rules that help to achieve long-enduring com-

mons [16]. Hardin’s famous phrase, “Tragedy of the Commons,” [17] describes settings where users who share a commons (e.g., a pasture) over-consume the resource, leading to its destruction. For each herdsman, the addition of one more animal to the herd adds positive utility because it is one more animal to sell. The negative is that it is one more animal grazing on the commons. The rational choice of each herder is to add more animals, leading eventually to overgrazing of the commons.

But because *Libre* software are digital, over-consumption of the commons is not the concern. Sustaining (and perhaps growing) a team of developers is. In these settings the tragedy to be avoided is the decision to abandon the project prematurely, not because of an external factor (such as a better technology has come along that is a better solution than what the project will produce), but because of some kind of problem internal to the project (such as a conflict over project direction, loss of financial support, etc.). (See Endnote 1.)

Charles M. Schweik is an Assistant Professor in the Dept. of Natural Resources Conservation and the Center for Public Policy and Administration at the University of Massachusetts, Amherst, USA. He has a PhD in Public Policy from Indiana University, a Masters in Public Administration from Syracuse University, and has an undergraduate degree in Computer Science. A primary research interest is in the use and management of public information technology. For more than six years, between his undergraduate degree and his MPA, he was a programmer with IBM. <cschweik@pubpol.umass.edu>

Since this is an important point, let me try and analyze this particular tragedy following Hardin's logic. In *Libre* software development settings, developers (and possibly users, testers, documenters) replace the herdsmen as decision-makers. The motivation for these people to participate is in part the anticipation that the software being produced will fill a personal or organizational need. However, research on *Libre* developer motivations [58] has shown that participants receive other positive benefits from participating.

From the developer's perspective, it is worth spending one unit of time contributing to this project because he is: (1) getting his name known and thus increasing the possibility for future job or consulting opportunities, (2) learning new skills through reading source and peer-review of their code submissions, and/or (3) getting paid by his employer to participate.

Alternatively, the logic might be taken to stop contributing time because he does not like the direction in which the project is going, or that his contributions are not being accepted and he is not receiving adequate feedback on why. In these situations, the accumulation of developer dissatisfaction may lead to premature project abandonment, because of factors internal to the project.

The tragedy of the commons in this context is about the premature loss of a production team, not over-appropriation as in Hardin's famous pasture example. Consequently, a key concern faced by Information Technology (IT) organizations who are considering *Libre* software as a policy or strategy is how can a vibrant production and maintenance effort be sustained over the longer term and how can the premature abandonment of the project be avoided.

In "Governing the Commons" [18], Elinor Ostrom emphasized that in some environmental commons settings Hardin's tragedy is avoided – the commons becomes "long-enduring" – because of the institutional designs created by self-governing communities. Institutions, in this context, can be defined as sets of rules – either formal or informal – consisting of monitoring, sanctioning and conflict resolution mechanisms that help to create appropriate incentive structures to manage the

commons setting. In *Libre* software commons settings, the evolution of project institutions may help to explain why some *Libre* software projects move more smoothly from alpha to beta to stable and regular release cycles and grow and maintain larger development teams and user communities, while other projects become abandoned before they reach maturity. While research shows that a vast majority of *Libre* software projects have either one developer or small teams [48-52], I think the influence of institutional design will become increasingly critical as projects grow (in terms of interested parties) and mature.

Moreover, the increasing involvement of firms and government agencies in *Libre* software development will undoubtedly lead to more complex institutional environments. For this reason, I think attention to the institutional designs of *Libre* projects is critically important as *Libre* software collaborations (and other "open content collaborations", see [13]) become more commonplace.

This paper describes some components of a five-year research program just underway which will study the institutional designs of *Libre* software development projects from a commons perspective. A primary goal of the research program is to identify design principles that contribute to the ultimate success or failure of these projects. The paper is organized in the following manner.

First, I explain why *Libre* software development projects are a type of commons, or more specifically, a "common property regime". Next, I define what we mean by institutions and describe a theoretical framework utilized by many social scientists to study institutional designs of environmental commons settings. I then describe the general trajectory of *Libre* software development projects, and discuss ways to measure success and failure at these stages of this trajectory. I provide some examples of hypotheses related to institutional designs that, when empirically tested, could help to identify design principles for *Libre* software projects. I close with a discussion of why this should matter to IT professionals.

| | | | |
|----------------------|-----------|------------------|-----------------------|
| | | RIVALROUS | |
| | | No | Yes |
| EXCLUDABILITY | Difficult | Public Goods | Common-Pool Resources |
| | Easy | Club Goods | Private Goods |

Figure 1: A General Clasification of Goods. (Adapted from [21:7].)

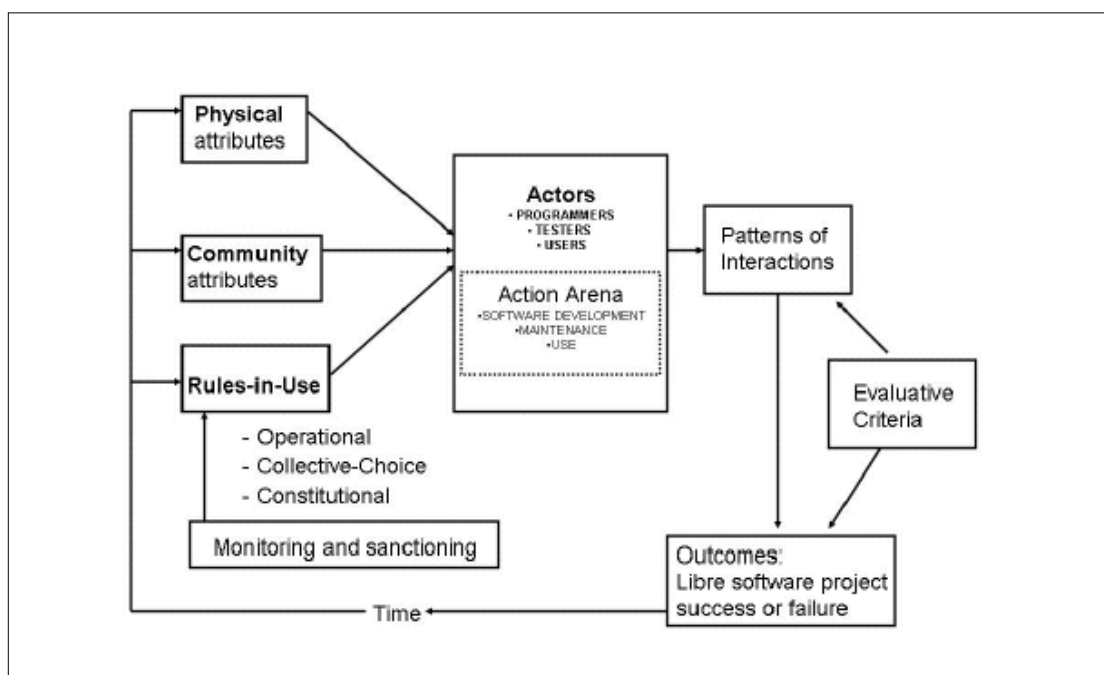


Figure 2: A Framework for Institutional Analysis of Commons Settings. (Adapted from [21:73].)

2 Libre Software Are Public Goods Developed by Common Property Regimes

It is possible to view *Libre* software from two perspectives: use and development. I'll consider the use side first. Social scientists recognize four categories of goods, private, public, club and common pool resources distinguished by two properties (Figure 1) [22][21]: first, how easy or difficult is it to exclude others from using or accessing the good? second, does the good exhibit rivalrous properties? That is, if I have one unit of the good, does that prevent others from using it as well?

Traditional proprietary software can be classified as a club good in Figure 1. The digital nature of software (downloadable over the Internet or copied off of CD-Rom) makes it non-rivalrous. The pricing for access (and the do-not-copy restrictions of the "shrinkwrap license") make exclusion of non-purchasers possible [53].

But in many cases, this exclusion is not always successful. It is widely understood that there illegal copying of proprietary software occurs, creating a different form of club with an entrance to the club based on the wiliness to risk being caught rather than on a price for access. But regardless of whether the company can successfully crack down on illegal bootlegging of their software, because of its digital nature, proprietary software falls in the club-good category.

Libre software differs from proprietary software in that *Libre* licenses (such as the GNU General Public

License) permit users to copy and distribute the software wherever they please as long as they comply with the specifications of the license [54]. These licenses provide a mechanism for acting upon a violation of the specified rules, so exclusion is theoretically possible through litigation under contract or copyright law [61-62], but in most cases this is unlikely [61].

Since *Libre* software is also non-rivalrous – it is freely copied digitally over the Internet or on CD-ROM (such as in the case of a Linux distribution, for example) – technically, it should be classified as a club good – a club with no "fee" other than license compliance to join. But given that *Libre* software distribution is global in reach with no monetary cost associated with it, many classify it a public good [23-25][25][55].

Now let me turn to a discussion about the production of *Libre* software. I noted earlier that some consider *Libre* software projects a form of "commons" [25][54]. McGowan [53] refers to these commons as "social spaces" for the production of freely available and modifiable software.

While these projects involve collaboration, and contrary to what some might believe, there are property rights (copyright) and ownership issues in these commons [53].

Raymond (cited by McGowan) defines owners of a *Libre* software project as ones who have "exclusive right, recognized by the community at large, to redistribute modified versions" [53: 24]. According to Raymond, one becomes owner of a project by either:

(1) being the person or group who started the project from scratch; (2) being someone who has received authority from the original owner to take the lead in future maintenance and development efforts; (3) being a person who takes over a project that is widely recognized as abandoned and makes a legitimate effort to locate the original author(s) and gets permission to take over ownership. McGowan adds a fourth option – the "hostile takeover" – where the project can be hijacked or "forked" because of the "new derivative works" permissions provided by the license. Forking often occurs when a project is deemed by some on the team to be headed technically or functionally in the wrong direction. A kind of mutiny can occur and a new project is created using the existing source from the old project. The result is two competing versions [53].

Some readers might find the definition of *Libre* project owners by Raymond somewhat troublesome. This definition encapsulates Raymond's libertarian view of *Libre* projects, where the community as a whole somehow together recognizes ownership rights and collectively acts as one to support them. To some, this collective recognition and action may appear rather hard to believe.

An alternative way of identifying or defining an owner in *Libre* software settings is through a person or team's ability to initiate or maintain a coherent collective development process.

From this perspective, ownership is more a result of the barriers against expropriation and does not require some mystical collective endorsement. The reader should note too that this alternative definition of *Libre* ownership is consistent with Raymond and McGowan's four ways to become a recognized *Libre* software owner listed above.

Given the ownership aspects above, here is the key point: *Libre* software projects are a form of self-governing "common property regime," with developers working collaboratively to produce a public good [9][11][12][27][13][25][53][54]. While 'commons' is the term most often used, "common property regime" more accurately describes *Libre* software projects.

The recognition of *Libre* projects as a form of common property regime provides an opportunity to connect knowledge amassed over the years on the governance and management of natural resource commons under common property settings (e.g., [14][15]).

Weber recently noted the importance of governance and management in *Libre* software projects when he stated: "*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*" [12:189].

3 A Framework for Studying The Institutional Designs of *Libre* Software Projects

Weber's recognition of social norms, informal coordination processes and formal governance structures coincides with what political scientists and economists refer to as "institutions" [18][21][31]. For more than 40 years, researchers, including this author [32-34], have utilized the "Framework for Institutional Analysis" (Figure 2) to organize thinking about environmental commons cases [31:8]. This framework has not yet been applied to the study of *Libre* software commons, but the analytic lens it provides complements other *Libre* software research underway by researchers in more traditional information systems fields (e.g., [35-38]).

Consider the situation where an analyst is trying to understand why a particular *Libre* software project is lively or why it is losing momentum. Figure 2 depicts *Libre* projects as a dynamic system with feedback. The analyst might begin studying the project by first looking to elements on the left-hand side: the physical, community and rule attributes.

Physical attributes refers to a variety of variables related to the software itself or to some of the infrastructure to coordinate the team. These include the type of programming language(s) used, the degree to which the software is modular in structure, and the type of communication and content management infrastructure used.

Community attributes refers to a set of variables relating to the people engaged in the *Libre* software project, such as whether they are volunteer or paid to participate, whether they all speak the same language or not, and aspects that are more difficult to measure related to social capital, such as how well team members get along, how well they trust each other [63], etc. This component also includes other non-physical attributes of the project, such as its financial situation and the sources that provide this funding (e.g., a foundation).

Rules-in-use refers to the types of rules in place that are intended to guide the behavior of participants as they engage in their day-to-day activities related to development, maintenance or use of the *Libre* software. The specific *Libre* license used is one important component of the rules-in-use category. But I expect that most *Libre* projects – especially more mature ones with larger numbers of participants – will have other sets of formal or informal rules or social norms in place that help to coordinate and manage the project.

The middle section of Figure 2, Actors and the Action Arena, indicates a moment or a range of time where the left side attributes remain relatively constant and actors (e.g., software developers, testers, users) involved in the *Libre* software project make decisions and take actions (e.g., programming, reviewing code, deciding to reduce or stop their participation, etc.). The aggregation of these actors making decisions and taking actions is depicted as Patterns of Interactions in Figure

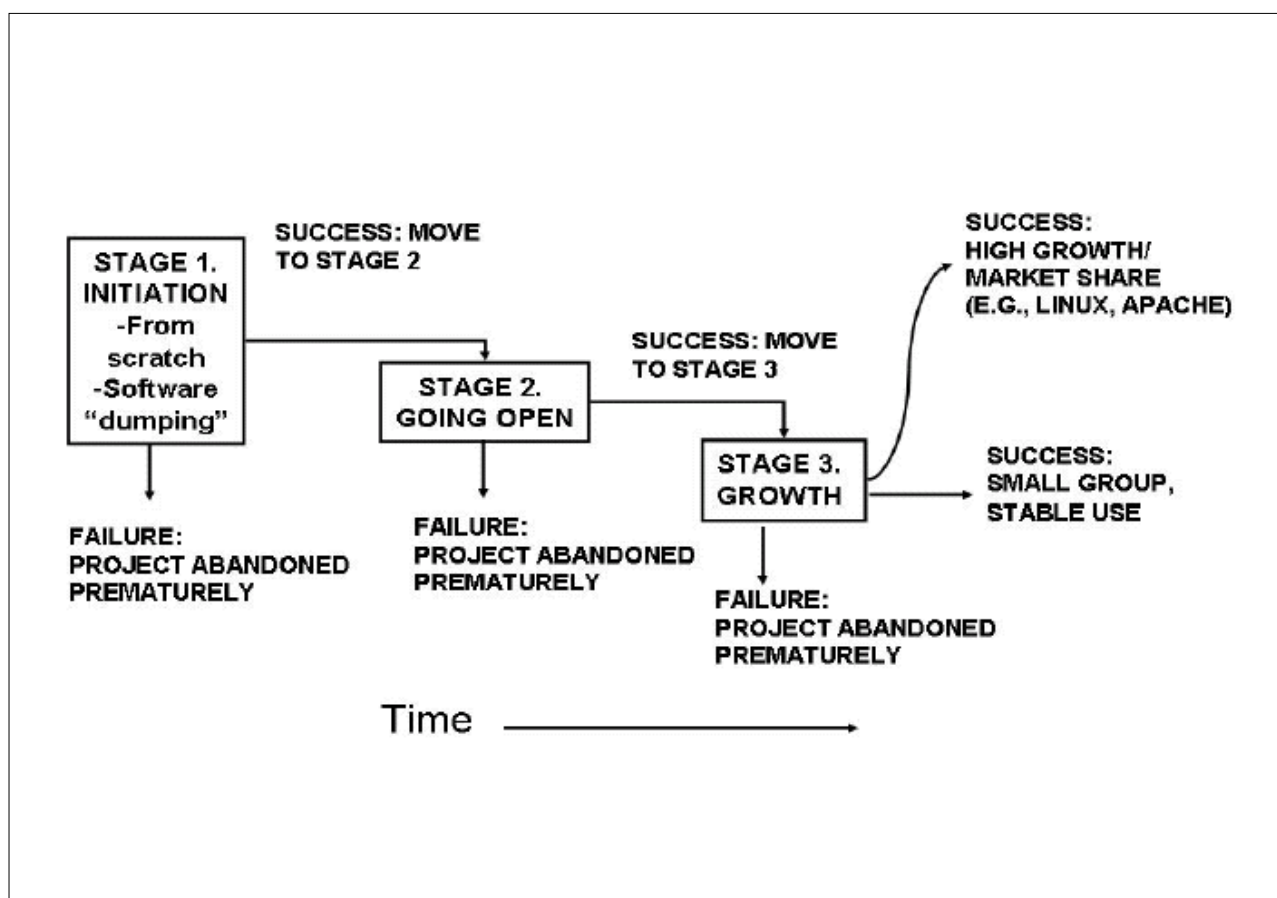


Figure 3: Stages of *Libre* Projects Software and Outcome (Success) Measures. (Adapted from Schweik and Semenov, 2003 [46].)

2. The accumulation of actions results in some Outcome (right side, bottom of Figure 2). An outcome could be a change in the physical attributes of the *Libre* software commons (e.g., a new release), a change in the community attributes of the project (e.g., new people joining in or people leaving), a change to the existing rules-in-use (e.g., a new system for resolving conflicts) or any combination thereof. In Figure 2, these kinds of changes are depicted through the feedback system from outcome to the three sets of commons attributes on the left side of the same figure, and a new time period begins.

Much of what institutional analysis is about is the study of rules, from formal laws to informal norms of behavior, standard operating procedures, and the like. Embedded in the rules-in-use category of Figure 2 are three nested levels that, together, influence actions taken and resultant outcomes at these different levels of analysis [39]. Operational level rules affect the day-to-day activities made by participants in the *Libre* software commons. These can be formally written rules, or, perhaps more often in *Libre* settings, could be community norms of behavior.

An example of operational rules might be the pro-

cedures for adding new functionality to the next release of the software. Another example might be rules on promoting a particular developer to a position where he or she has more decision-making responsibility in the project. The Collective-Choice level represents the discussion space where team members with authority define group goals, and create or revise operational rules to move the team toward these goals. In addition, at this level there is a system of collective-choice rules that define who is eligible to change operational rules and specify the process for changing these rules [21]. Collective-choice rules might, for example, determine how the team can change the process of code review before new code can be checked-in or "committed" [40]. Constitutional-Choice level rules specify who is eligible to change collective-choice rules and also define the procedures for making such changes. For example, if the formally designated leader of a *Libre* project has decided to move on to a new opportunity, constitutional-choice rules would outline how a replacement is chosen.

At each level, there can be systems for the monitoring of rule conformance and systems for sanctioning when rules are broken.

In short, at any point in time in the lifecycle of a *Libre* software project, programmers, users, and testers will make their decisions on how they participate based on the existing physical, community and institutional attributes of the project, as well as their anticipation of where the project is headed and their own personal circumstances. Participants make decisions and take actions at three levels: operational, collective-choice and, less frequently, constitutional-choice.

One hypothesis to be tested in this research is that the systems of rules-in-use at any one of these levels will become more complex as a *Libre* software project matures and gains participants. I also expect the institutional design to become more complex in situations where one or more organizations (e.g., firms, non-profits or government agencies) provide resources to support the project. This is consistent with McGowan [53:5] when he states: "*The social structures necessary to support production of large, complex projects are different from the structures – if any – necessary to support small projects...*".

4 The Trajectory of *Libre* Software Projects

I now turn to the question of how to evaluate the "Outcomes" component in this framework. While Figure 2 reveals a feedback loop to draw attention to the dynamic and evolutionary nature of these projects [48][45], it doesn't depict these longitudinal properties very well. For this reason I include Figure 3.

In earlier work [46] I argued that *Libre* software projects follow a three-stage trajectory (Figure 3): (1) initiation; (2) a decision to "go open" and license it as *Libre* software; and (3) a period of growth, stability or abandonment. Most of *Libre* software research focuses on projects at Stage 3. But some of the decisions made at the earlier stages may be critical factors leading to the outcome of growth or abandonment at Stage 3.

Consider Stage 1 in Figure 3. In many cases, *Libre* software projects start with a private consultation of one or a few programmers working together to develop a 'core' piece of software.

At this juncture, the software may not yet be placed under a *Libre* software license or made available on the Internet, and in some circumstances the team may not even be planning at the moment to license it as *Libre* software. But critical design decisions may be made at this stage, such as the modularity of the code, which might greatly influence how well the software can be developed in a *Libre* common property setting in Stage 3.

While the "small and private group starting from scratch" scenario is probably what most might think of in the initiation phase of a *Libre* software project, there is at least one other alternative: "software dumping" [60]. In these situations, the software first is developed under a more traditional, proprietary, closed-source, software-development model within a firm. At some

point – perhaps after years of work – decision-makers may make a strategic decision not to support the software any more, and as a consequence, make the code available and license it as *Libre* software. This scenario may become more prominent in future years if software firms continue to consider *Libre* software as part of their business strategy.

The "going open" stage (Figure 3, Stage 2) is probably brief but perhaps not as simple as it might at first appear. In this stage, team members decide on an appropriate *Libre* software license, and, perhaps more importantly, create a public workspace and a collaborative infrastructure (e.g., versioning system, methods for peer review, bug tracking mechanism, etc.) to support the project.

Platforms like Sourceforge.net and Freshmeat.net have made this step fairly easy, but there are some projects that utilize web-based platforms that they have implemented themselves.

I should note at this juncture that in some *Libre* projects, Stages 1 and 2 can be conflated. It may be a relatively common phenomenon where a founding member has an idea and immediately broadcasts an appeal for other partners to help in the development of the project.

This appeal may immediately involve the creation of a project page on the web or on a hosting site such as Sourceforge.net. But regardless of how the project gets through Stage 2, the next step is Stage 3 of Figure 3. This stage describes the period in the project's life cycle where the software is actively being developed and used under *Libre* software licensing and is publicly available on the Internet. Many of the early studies of *Libre* software projects focused on cases that fall under the "high growth" (in terms of market share or number of developers) success stories such as Linux or Apache Web Server. Achieving this stature is often the default or assumed measure of success of these projects in *Libre* software literature.

However, empirical studies of *Libre* software have shown that the majority of projects never reach this stage and many, perhaps most, involve only a small number of individuals [48-52]. Some of these studies may be focusing on projects in the early days of their life cycle, where people are working to achieve high growth. But in other instances, members of a particular project may be quite satisfied to remain "stable" and active with a small participant base (Figure 3, Stage 3: Small Group). Some *Libre* software projects in bioinformatics might provide examples of these kinds of circumstances [47].

The main point regarding Figure 3 is that there are important stages in the trajectory of *Libre* software projects and that the measures for success and failure will likely change during these stages. Moreover, physical, community and institutional attributes of projects will evolve as well.

5 Measuring Success or Failure of Libre Software Projects along This Trajectory

I noted earlier that a goal of this research project is to define "design principles" that lead to successful *Libre* software collaborations. In the empirical work I am just initiating, success or failure of *Libre* projects is the concept I seek to explain. What follows is a description of one method for measuring success and failure. Others have undertaken research trying to quantify this as well, and I build upon their important work [3][4][8][41].

For my purposes, an initial measure of success or failure in *Libre* project collaboration requires asking two questions in sequence. First, does the project exhibit some degree of development activity or from a development perspective does the project look abandoned? Second, for projects that appear to be abandoned, were they abandoned for reasons that were outside of the team's control? Let me elaborate on each question.

5.1 Does The Project Exhibit Developer Activity or Does It Look Abandoned?

Several studies have measured whether a *Libre* software project is 'alive' or 'dead' [48], by monitoring changes in the following physical attribute variables of the software (Figure 2) over some period of time:

- Release trajectory (e.g., movement from alpha to beta to stable release) [3].
- Version number [3][48].
- Lines of code [48][43].
- Number of 'commits' or check-ins to a central storage repository [45].

Similarly, the analyst could monitor changes in community attribute variables (Figure 2) such as:

- The activity or vitality scores as measured on collaborative platforms such as Sourceforge.net or Freshmeat.net [3][8][48].

Obviously, if any of these metrics show some change over a period of time, the project demonstrates some level of activity or life. A key issue here will be deciding what time range will be long enough to determine a dead or abandoned project. I expect that some more mature software projects may show periods of dormancy in terms of development activity until some interesting new idea is suggested by a developer or user. Consequently, the range of time with no signs of activity should be relatively long in order to determine project death, or, better yet, the analyst should find some acknowledgment in project documentation (e.g., website) that the project has been closed down or abandoned.

5.2 If The Project Looks Abandoned, Did The Abandonment Occur because of Factors Outside of The Team's Control?

A classification of a project as dead does not by

itself necessarily mean it was a failed project [63]. Some projects may exhibit no activity because they have reached full development maturity: the software produced does the job and requires no more improvements. In these instances, the project would be classified as a success story, not a failure.

In other instances a project may be classified as dead or abandoned but have become so for reasons outside the project team's control, such as in the case where another (rival) technology has come along that is technologically superior or becomes more accepted (recall the Gopher versus WWW example in Endnote 1). In these instances, the project should probably not be considered a failure from a collaborative point of view (although in some cases they probably could be). There were simply rival technologies that were better at doing the job.

But there will be other cases where the project shows no signs of development life and yet the software has not fully matured to its full potential and there were no apparent external factors that led to developers abandoning the project. I would classify these as premature abandonment cases, for some factor in the internal project led to people abandoning the effort before it reached maturity.

Consequently, for this research program, I intend to use the questions in Section 5.1 and 5.2 to classify projects into success or failed categories. Successful projects will either show some level of life or will exhibit no further signs of development because it has reached development maturity.² Projects that were abandoned because of external influences will be dropped out of the population of cases, for they cannot be classified as either a success story or a failed case. Cases that appear to be abandoned prematurely because of some kind of internal issue or issues will be the ones classified as failures. These metrics will be fairly easy to collect for projects that are in Stage 3 (growth) in Figure 3. It will be more difficult to identify projects to study that fall in the earlier stages (Stage 1 or 2), but when I do, these same concepts should apply.

6 Toward The Identification of "Design Principles" for Libre Software Commons

Until now, I have tried to emphasize four points. First, that *Libre* software projects are a form of common property regime which have physical, community and institutional attributes that influence their performance. Second, that there are many ways to measure the success and failure of these projects but an important one will be a measure of the collaboration being abandoned prematurely (failure) or remaining until the software reaches a level of maturity (success). Third, that the institutional designs – the rules-in-use – are an area that has to date been largely neglected in *Libre* software studies. Fourth, that the identification of "design principles" which lead to success of these projects at the

different stages in Figure 3 is desirable as more organizations turn to *Libre* software as an IT strategy.

The identification of design principles will require a systematic study of *Libre* software projects at different stages in Figure 3 with attention given to appropriate measures of success and failure at each stage. Hypotheses will need to be made related to all three sets of independent variables – physical, community and institutional attributes in Figure 1 – based upon work in traditional software development, more recent studies of *Libre* software projects explicitly, and relevant work related to natural resource commons. But because of space limitations, I will conclude this paper by providing some hypotheses related to the institutional designs of *Libre* software projects (rules-in-use in Figure 1) and noting their relationship to studies of natural resource commons.

Libre software projects will be more successful (not be abandoned prematurely) if they include some degree of voice by lower level participants in the crafting of operational-level rules.

It has been shown that in natural resource commons settings the resources are better sustained when users have some rights to make and enforce their own operational-level rules [18][14]. Applying this to *Libre* software projects if, for example, operational-level rules are imposed by some overarching authority without the consultation of others working "in the trenches," these workers may become disenchanted and abandon the project. Alternatively, if developers and users associated with the *Libre* software project have some say in the crafting or revising of operational-level rules as the project progresses, commons theory suggests they will be more willing to participate over the long run.

Libre software projects will be more successful (not be abandoned prematurely) if they have established collective-choice arrangements for changing operational rules when necessary.

It has also been shown that long-enduring natural resource commons tend to have institutional designs that allow for rule adaptation when needed. Systems with fixed rules will more likely fail because the understanding at the time they were crafted may be, to some degree, flawed or the situation they were designed to work in will eventually, change [15].

Libre software projects will be more successful (not be abandoned prematurely) if they have systems established to handle disputes between group members.

Studies such as Divitni et al. [56] and Shaikh and Cornford [57] provide discussions on conflict in *Libre* software settings. The most extreme type of conflict is "forking," described earlier. Commons settings with conflict management in place often result in early resolution coupled with new learning, and understanding within the group [15:1909]. Projects not capable of handling conflict can lead to dysfunctional situations where cooperation is no longer possible.

Libre software projects will be more successful (not be abandoned prematurely) if

... they have systems in place that provide for the monitoring of operational rules.

... they have some level of graduated sanctions for people who break established rules.

... they have rule enforcers whose judgments are deemed effective and legitimate.

Operational rules work only if they are enforced. Research in natural resource commons settings has shown that often systems of low-cost monitoring can be established by the users themselves, and are most effective when there are (at first) modest sanctions given to offenders [15]. Sharma, Sugumaran and Rajgopalan [59] note that monitoring and sanctioning systems do exist in some form in *Libre* software projects. However, in current *Libre* software literature little is mentioned on this topic. Commons literature suggests that the chance for success will be higher if there is formal or informal monitoring of operational rule conformance as well as a set of tiered social sanctions in place to rein in rule-breakers. For example, a remonstrance by direct one-to-one email that, if not successful, progresses to "flaming" in view of the entire team is one example of a graduated sanction procedure in *Libre* software settings. In addition, commons studies have also shown that rule enforcement is more apt to work when the people imposing sanctions are deemed effective and legitimate [15]. Translated to *Libre* cases, effective sanctioning of rule breakers requires someone who possesses formal designation to do this or who is recognized as a legitimate group authority.

7 Conclusions

The hypotheses provided in the previous section are intended to be illustrative of what needs to be done to move toward the identification of design principles in *Libre* software commons settings. I have provided examples highlighting institutional (rules-in-use) issues because I think this is an area that has, until now, been neglected in the *Libre* software research. However, testable hypotheses certainly can be generated related to other categories of attributes on the left side of Figure 1. For example, an obvious but important one probably is: *Libre* software projects will be more successful if they have a regular and committed stream of funding coming in to support their endeavor.

It may be that, for many *Libre* projects, attention to institutional design is simply not important, because the development team is comprised of only one or a small number of individuals. More important variables at that stage may be physical or community attributes. However, I suspect that in the larger (in terms of lines of code) projects, or in *Libre* projects where more than one firm or organization is contributing resources to support the project, the institutional design will become a much more important set of variables. Over the next few years, funded by the U.S. National Science Foundation, I will be undertak-

ing a systematic study of these projects looking specifically at the design and evolution of their institutional structures and these issues.

Why should **UPGRADE** and **Novática** readers care? Here I return to where I started. Several papers in their 2001 issue emphasized the changing nature of participation in *Libre* software projects: that increasingly actors are not volunteers but people paid by their organizations to contribute to the development of the software. It is not difficult to imagine a future where government agencies and/or firms devote resources to work on a *Libre* software project together. (Firms are doing this as right now). A main lesson from natural resource commons research is that institutions matter. I expect that as *Libre* software and *Libre* software commons mature, institutional attributes will become increasingly important and apparent as factors that lead to the success or failure of these projects.

Endnotes

1. I am indebted to an anonymous reviewer for making the point that some abandoned projects are not tragedies. This reviewer provided the example of the Gopher technology being superseded by the World Wide Web technology. This is a case of an external factor leading to the early abandonment of the software project but would not be considered a tragedy. I should also note that the idea of project cancellation has been used in more traditional software development in the past [42], but the phrase "premature abandonment" rather than "premature cancellation" better fits *Libre* settings since in many cases there is no formal organization making the decision to end the project prematurely.

2. An additional analytic part of this project will be to analyze the 'vibrancy' of successful projects – capturing the degree of life (in terms of developer or user activity) a project exhibits. In other words, I ultimately want to develop a measure of success that moves beyond the 'live' versus 'dead' metric. Several studies (e.g., [3][4][8]) have looked at vibrancy metrics, focusing in on variables such as the numbers of people in the formal development team or the extended development team (e.g., bug reporters), number of commits, number of downloads, etc. Other possible vibrancy metrics might include an examination of the direction of change in numbers of formal or extended developer teams. However, a more thorough examination of these metrics is needed – beyond what can be done in this paper. For it is likely that any vibrancy metric will be closely tied to the stage of development of the project. For example, Dalle and colleagues [44] note that more active, younger projects on Sourceforge.net are likely to attract developers at a higher rate than older, more mature projects with larger code bases. From this perspective, vibrancy metrics might look very similar between a project that is being abandoned prematurely and a project that is reaching maturity. For this reason,

in this paper I only want to point out that I intend to investigate further how to conceptualize and put into operation vibrancy metrics but it is beyond the scope of this paper to do so.

Acknowledgments

Support for this study was provided by a grant from the U.S. National Science Foundation (NSFIS 0447623). However, the findings, recommendations, and opinions expressed are those of the authors and do not necessarily reflect the views of the funding agency.

References

- [1] UPGRADE, Vol. II, No. 6, December 2001, <<http://www.upgrade-cepis.org/issues/2001/6/upgrade-vII-6.html>>; Novática, n. 154 (nov.-dic. 2001), <<http://www.ati.es/novatica/2001/154/nv154sum.html>> (in Spanish).
- [2] R.W. Hahn. "Government Policy Toward Open Source Software: An Overview." In R. W. Hahn (ed.) Government Policy toward Open Source Software. Washington, D.C.: AEI-Brookings Joint Center for Regulatory Studies, 2002.
- [3] K. Crowston, H. Annabi, and J. Howison. "Defining Open Source Project Success." In Proceedings of the 24th International Conference on Information Systems (ICIS 2003). Seattle, WA, 2003.
- [4] Crowston, Annabi, Howison, and Masango. "Towards a Portfolio of FLOSS Project Success Measures." In Feller, Fitzgerald, Hissam, and Lakhani (eds.) Collaboration, Conflict and Control: The Proceedings of the 4th Workshop on Open Source Software Engineering. Edinburg, Scotland, 2004.
- [5] K. Crowston and B. Scozzi. "Open Source Software Projects as Virtual Organizations: Competency Rallying for Software Development," IEE Proceedings Software (149:1), pp. 3-17, 2002.
- [6] W. Scacchi. "Understanding the Requirements for Developing Open Source Software Systems." IEE Proceedings Software. 149, 1: 24-39, 2002.
- [7] W. Scacchi. "Free and Open Source Development Practices in the Game Community." IEEE Software. January/February, 2004.
- [8] K.J. Stewart and T. Ammeter. "An Exploratory Study of Factors Influencing the Level of Vitality and Popularity of Open Source Projects." In L. Applegate, R. Galliers, and J.I. DeGross (eds.) Proceedings of the Twenty-Third International Conference on Information Systems, Barcelona. Pp. 853-57, 2002.
- [9] Y. Benkler. "Coase's Penguin, or Linux and the Nature of the Firm. Yale Law Journal. 112 (3), 2002.
- [10] A. Nuvolari. "Open Source Software Development: Some Historical Perspectives." Eindoven Center for Innovation Studies, Working paper 03.01. <<http://opensource.mit.edu/nuvolari.pdf>>, 2003.
- [11] J. Boyle. The second enclosure movement and the construction of the public domain. Law and Contemporary Problems. 66(1-2): 33-75, 2003.
- [12] S. Weber. The Success of Open Source. Cambridge, MA: Harvard University Press, 2004.
- [13] C.M. Schweik, T. Evans, and J.M. Grove. "Open Source and Open Content: A Framework for the Development of Social-Ecological Research." Ecology and Society (pending publication).
- [14] E. Ostrom, J. Burger, C.B. Field, R.B. Norgaard, and D.

- Policansky. "Revisiting the Commons: Local Lessons, Global Challenges." *Science* 284. pp. 278-282, 1999.
- [15] T. Dietz, E. Ostrom, and P. Stern. "The Struggle to Govern the Commons." *Science* 302(5652). pp. 1907-1912, 2003.
- [16] C. Hess and E. Ostrom. "Ideas, Artifacts and Facilities: Information as a Common-Pool Resource. *Law and Contemporary Problems*. 66(1&2), 2003.
- [17] G. Hardin. "The Tragedy of the Commons." *Science*. 162:1243-48, 1968.
- [18] E. Ostrom. *Governing the Commons: The Evolution of Institutions for Collective Action*. New York: Cambridge University Press, 1990.
- [19] R. Netting. *Balancing on an Alp: Ecological Change and Continuity in a Swiss Mountain Community*. Cambridge: Cambridge University Press, 1981.
- [20] J.M. Baland and J.P. Platteau. *Halting Degradation of Natural Resources. Is there a Role for Rural Communities?* Oxford University Press, 1996.
- [21] E. Ostrom, R. Gardner, and J.K. Walker. *Rules, Games, and Common-Pool Resources*, Ann Arbor: University of Michigan Press, 1994.
- [22] V. Ostrom and E. Ostrom. "Public Goods and Public Choices." In *Alternatives for Delivering Public Services: Toward Improved Performance*. E.S. Savas (editor). Boulder, Colo: Westview Press. pp. 7-49, 1977.
- [23] J. Bessen. "Open Source Software: Free Provision of Complex Public Goods." <<http://www.researchoninnovation.org/opensrc.pdf>>, 2001.
- [24] Peter Kollock. "The Economies of Online Cooperation: Gifts and Public Goods in Computer Communities." In *Communities in Cyberspace*, edited by Marc Smith and Peter Kollock. London: Routledge, 1999.
- [25] R. van Wendel de Joode, J.A. de Bruijin, and M.J.G. van Eeten. *Protecting the Virtual Commons: Self Organizing Open Source and Free Software Communities and Innovative Intellectual Property Regimes*. The Hague: T.M.C. Asser Press, 2003.
- [26] S.V. Ciriacy-Wantrup and R.C. Bishop. "'Common Property' as a Concept in Natural Resource Policy." *Natural Resources Journal*. 15:713-27, 1975.
- [27] A. Nuvolari. "Open Source Software Development: Some Historical Perspectives." Eindhoven Center for Innovation Studies, Working paper 03.01. <<http://opensource.mit.edu/papers/nuvolari.pdf>>, 2003.
- [28] K. Healy and A. Schussman. "The Ecology of Open-Source Software Development." Available at <<http://opensource.mit.edu/papers/healyschussman.pdf>>, 2003.
- [29] B.J. McCay and J.M. Acheson. *The Question of the Commons: The Culture and Ecology of Communal Resources*. Tucson: University of Arizona Press, 1987.
- [30] D. W. Bromley et al. *Making the Commons Work: Theory, Practice, and Policy* (ICS Press, San Francisco, 1992).
- [31] C. Hess and E. Ostrom. "A Framework for Analyzing Scholarly Communication as a Commons." Presented at the Workshop on Scholarly Communication as a Commons, Workshop in Political Theory and Policy Analysis, Indiana University, Bloomington, IN, March 31-April 2, 2004. <<http://dlc.dlib.indiana.edu/archive/00001244/>>.
- [32] C.M. Schweik. *The Spatial and Temporal Analysis of Forest Resources and Institutions*. Tesiks Doctoral. Center for the Study of Institutions, Population and Environmental Change, Indiana University. Bloomington, IN, 1998.
- [33] C.M. Schweik. *Optimal Foraging, Institutions and Forest Change: A Case from Nepal*. *Environmental Monitoring and Assessment*. 62: 231-260, 1999.
- [34] Schweik, C.M., Adhikari, K., and Pandit, K.N. 1997. "Land-Cover Change and Forest Institutions: A Comparison of Two Sub-Basins in the Southern Siwalik Hills of Nepal." *Mountain Research and Development*. 17(2): 99-116, 1997.
- [35] Institute for Software Research. <<http://www.isr.uci.edu/research-open-source.html>>, 2005.
- [36] Libre Software Engineering. <<http://libresoft.urjc.es/>>, 2005.
- [37] FLOSS, Free/Libre Open Source Software Research. <<http://floss.syr.edu/>>, 2005.
- [38] K. Stewart. Open Source Software Development Research Project. <<http://www.rhsmith.umd.edu/faculty/kstewart/ResearchInfo/KJSResearch.htm>>, 2005.
- [39] L.L. Kiser and E. Ostrom. "The Three Worlds of Action: A Meta-theoretical Synthesis of Institutional Approaches." In E. Ostrom (ed.) *Strategies of Political Inquiry*. Beverly Hills, CA: Sage. Pp. 179-222, 1982.
- [40] K. Fogel and M. Bar. *Open Source Development with CVS*. Scottsdale, AZ: Coriolis, 2001.
- [41] K. Stewart. "OSS Project Success: From Internal Dynamics to External Impact." In *Proceedings of the 4th Annual Workshop on Open Source Software Engineering*. Edinburgh, Scotland. May 25th, 2004.
- [42] Standish Group International, Inc. *The CHAOS Report*. <http://www.standishgroup.com/sample_research/chaos_1994_1.php>, 1994.
- [43] S. Hissam, C.B. Weinstock, D. Plaksoh, and J. Asundi. *Perspectives on Open Source Software*. Technical report CMU/SEI-2001-TR-019, Carnegie Mellon University. <<http://www.sei.cmu.edu/publications/documents/01.reports/01tr019.html>>, 2001.
- [44] J-M. Dalle, P.A. David, R.A. Ghosh, and F.A. Wolak. "Free & Open Source Software Developers and 'the Economy of Regard': Participation and Code-Signing in the Modules of the Linux Kernel." <http://siepr.stanford.edu/programs/OpenSourceDavid/Economy-of-Regard_8+_OWLS.pdf>, 2004.
- [45] G. Robles-Martinez, JM. Gonzalez-Barahona, J. Centeno-Gonzalez, V. Matellan-Olivera, and L. Rodero-Merino. "Studying the Evolution of Libre Software Projects Using Publically Available Data. In J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani (eds.) *Taking Stock of the Bazaar: Proceedings of the 3rd Workshop on Open Source Software Engineering*. <<http://opensource.ucc.ie/icse2003>>, 2003.
- [46] C.M. Schweik and A. Semenov. *The Institutional Design of "Open Source" Programming: Implications for Addressing Complex Public Policy and Management Problems*. *First Monday* 8(1). <http://www.firstmonday.org/issues/issue8_1/schweik/>, 2003.
- [47] <<http://bioinformatics.org/>>.
- [48] A. Capiluppi, P. Lago, and M. Morisio. "Evidences in the Evolution of OS projects through Changelog Analyses." In J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani (eds.) *Taking Stock of the Bazaar: Proceedings of the 3rd Workshop on Open Source Software Engineering*. <<http://opensource.ucc.ie/icse2003>>, 2003.
- [49] R.A. Ghosh and V.V. Prakash. "The Orbiten Free Software Survey." *First Monday* (5) 7. <http://firstmonday.org/issues/issue5_7/ghosh/>, 2000.
- [50] R.A. Ghosh, G. Robles, and R. Glott. *Free/Libre and Open Source Software: Survey and Study*. Technical report. International Institute of Infonomics. University of Maastricht,

- The Netherlands. June. <<http://www.infonomics.nl/FLOSS/report/index.htm>>, 2002.
- [51] K. Healy and A. Schussman. "The Ecology of Open-Source Software Development." Disponible en: <<http://opensource.mit.edu/papers/healyschussman.pdf>>, 2003.
- [52] S. Krishnamurthy. "Cave or Community? An Empirical Examination of 100 Mature Open Source Projects." *FirstMonday* 7, 2002.
- [53] D. McGowan. *Legal Implications of Open Source Software*. University of Illinois Review, 241 (1): 241-304, 2001.
- [54] E. Moglen. "Anarchism Triumphant: Free Software and the Death of Copyright." *First Monday*, 4. August, 1999.
- [55] P. Kollock. *The Economies of Online Cooperation: Gifts and Goods in Cyberspace*. In M. Smith and P. Kollock (eds.) *Communities in Cyberspace*. London: Routledge. Pp. 220-239, 1999.
- [56] M. Divitini, L. Jaccheri, E. Monteiro, and H. Traetteberg. "Open Source Processes: No Place for Politics?". In J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani (eds.) *Taking Stock of the Bazaar: Proceedings of the 3rd Workshop on Open Source Software Engineering*. <<http://opensource.ucc.ie/icse2003>>, 2003.
- [57] M. Shaikh and T. Cornford. "Version Management Tools: CVS to BK in the Linux Kernel." In J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani (eds.) *Taking Stock of the Bazaar: Proceedings of the 3rd Workshop on Open Source Software Engineering*. <<http://opensource.ucc.ie/icse2003>>, 2003.
- [58] J. Feller and B. Fitzgerald. *Understanding Open Source Software Development*. London: Addison Wesley, 2002.
- [59] Sharma, Sugmaran, and Rajgopalan. "A Framework for Creating Hybrid-Open Source Software Communities." *Information Systems Journal*. 12:7-25, 2002.
- [60] A. Wasserman, A. Center for Open Source Innovation, Carnegie Mellon West Coast Campus. Personal conversation.
- [61] L. Rosen. *Open Source Licensing*. Upper Saddle River, NJ: Prentice Hall, 2005.
- [62] A.M. St. Laurent. *Understanding Open Source and Free Software Licensing*. Sebastopol, CA: O'Reilly, 2004.
- [63] J.M. Garcia. "Quantitative Analysis of the Structure and Dynamics of the Sourceforge Project and Developer Populations: Prospective Research Themes and Methodologies." <http://siepr.stanford.edu/programs/OpenSoftware_David/JuanM-G_FOSS-PopDyn_Report2+.pdf>, 2004.