An Institutional Analysis on Open Source: How Developers Use Knowledge and Entrepreneurship in Communities

A thesis presented to the faculty of CEVRO
in partial fulfillment for the degree of
Master of Political Science, Philosophy, and Economics

CEVRO INSTITUTE
Prague, Czech Republic

Author
Don LIM

Advisor
Dr. Adam MARTIN

April 29, 2019

Abstract

The emergence of open source (OS) software development has presented some counterintuitive economic thinking. The current mainstream consensus is that monetary incentives and strongly enforced property rights encourage innovation and development. OS seemingly provides little monetary incentives and the benefits from providing software are difficult if not impossible the internalize. An encompassing overview of the open source literature will be conducted along with an attempt to answer the fundamental question which remains: why developers dedicate their time working for free in OS rather than earning income while developing within firms. This thesis presents and attempts to use Professor Elinor Ostrom's Institutional Analysis and Development (IAD) framework, Professor F.A. Hayek's work on spontaneous and emergent order, and entrepreneurship theory (focusing on the work done by Professor Peter G. Klein, Professor Nicolai Foss, and Professor Israel Kirzner) to help model the phenomenon of OS: the community and social institutions, how knowledge is generated and used, the motivational influences for voluntary contributions, and how developers inititate useful projects. The changes in private property regimes relating to hierarchical structures, licensing schemes, and community ownership will also be analyzed. Lastly, common arguments for and against government intervention, regulation, and promotion of OS will be examined. Although OS has grown significantly the past few decades, it is unlikely to overtake traditional firms; just as there is a coexistence between public and private ownership in markets and firms. Proprietary and OS development will continue to compete and cooperate in interesting and emergent ways.

Acknowledgments

First and foremost, I would like to thank my advisor Professor Adam Martin for thanking his time and effort giving me organizational advice, directing my inquiries, and giving me great book and paper recommendations.

For providing assistance and advice on this thesis, I would like to thank the Free Market Institute at Texas Tech University for allowing me to partake in courses, events, and social gatherings which helped expose me to new ideas, methods, and avenues of thought. In particular, I am grateful for the helpful comments provided by Professor Benajmin Powell and Professor Alexander Salter.

Next, I would like to thank the American Institute for Economic Research for hosting me and providing housing, library access, funding, and advice for my thesis. The entire facility is beautiful, the people remarkable, and the entire atmosphere conducive to objective research. I would like to thank Professor Edward Stringham for making this opportunity possible.

Of course, I would like to thank my home university, the CEVRO Institute, for providing the opportunity and the funding for my study in Prague with a faculty of prestigious and internationally recognized professors. I am extremely grateful for Professor and Director Josef Šíma for his advice, his work, and his care in the PPE program. Professor Michael Munger, Professor Mark LeBar, and Professor Peter Boettke has been and continue to be personally motivational and influential in my thinking and in my pursuit of further knowledge. Their aid in founding the PPE program has provided me—and I suspect the rest of the class—with immeasurable experiences, ideas, and opportunities. While I cannot name the entire faculty and all the visiting professors which influenced me, presently or in the future when I will most likely revisit their ideas, I would like to express my gratitude to Professor Judit Kapas for her course on *Entrepreneur and the Firm* and Professor Boudewijn Bouckaert and Professor Katarina Zajc for their course on *Law and Economics of Property Rights*. The numerous and interesting discussions provided by my fellow students in the program will forever and always influence me—which will be eternally treasured.

Lastly but not leastly, I would like to thank my family for their never-ending support.

Contents

I. Introduction

Open source (OS) poses a few problems to classical economic theory. Two problems are immediately evident: (1) why do talented individuals develop OS software instead of earning income by developing in a proprietary (or closed) manner through firms;[1] and (2) since OS software is often considered a public good[2]—though some argue a privately provided public good (Hippel & Krogh, 2003; J. P. Johnson, 2002)—there should remain the associated issues of free-riding (Olson, 1965/1971), motivation, and tragedy of the commons (Hardin, 1968). Paradoxically, contrary to economic theory, despite most OS software is produced with little to no charge in price, the reputation of OS software remains that of high quality, particularly for the more popular OS projects. OS software further challenges the conventional view that innovation is best supported by strong property rights (Demsetz, 1970). Demsetz found that if the benefits of privatizing and selling goods or services cannot be easily internalized, property rights will be too costly to emerge. OS development does the opposite; it attempts to externalize most benefits. The role of transaction costs, how much easier communication and widespread hardware is, and how OS attempts to further decrease these costs will be explored throughout. The price mechanism and its signaling capabilities seems weaker in OS than through the market or through firms. The foremost question explored in this paper is, why OS succeeds with this limited reliance on the price mechanism.

The demand for increasingly faster and more powerful technology, hardware and software, and rapid innovative growth has led many individuals and companies investing in and creating proprietary software. Proprietary software is protected by intellectual property rights—by copyright, trademark, registered design, or patent. As a result, only certain persons, teams, or organizations with these rights are allowed to read, develop, and maintain

1 A question first posited by J. Lerner and Tirole (2002).

2 The theory of public goods remains contested since Samuelson (1954) first defined and analyzed it as collective consumption goods, though similarly, A. Smith (1776/1998, p. 915) and McCulloch (1864/2013, p. 39) have also claimed certain goods cannot be provided for through voluntary collective action. Buchanan (1965), Cowen (1985), and Holcombe (1997) found the theory of public goods inadequate and inappropriate to justify public expenditure. Common claims for public goods involving science and education is further contested by Kealey (1996) and Polanyi (1951/1969) and Shaw (2010), respectively. However, rather than debate the validity of the theory of public good in this paper, the relevance of non-rivalrous and non-excludable goods (Cornes & Sandler, 1994) is assumed. Although OS software is different than other public goods it is, however, by definition non-exclusionary and is a good which does not diminish when consumed by another individual (Snidal, 1979).

the code. It is difficult to imagine developing software by any other method since the process is costly, expertise is required, and the organizational structure required is complex; yet open source software exists and thrives. The emergence of OS hints at a "new intellectual property paradigm" (Maurer & Scotchmer, 2006), though Foss (2002) remained skeptical and critical of these claims about OS and the overall emerging knowledge economy. However, rather than slowing down, there has been an exponential growth in OS development (Deshpande & Riehle, 2008).

OS software offers source code that is open to public inspection, distribution, and modification. The nature of OS software is to allow easy access to source code, code repositories, contributors, and other project data. Many of the features and modifications in OS projects are due to voluntary submissions—often from first-time contributors, which may consist of commits,[3] fixing typos or bugs, or commenting on an issue. Thousands will make contributions to the body of knowledge while core developers (the terms *developer* and *programmer* in this paper will be used interchangeably) will also contribute and make the final decisions in which code to incorporate into the project. Developers are those who write, maintain, reuse, or in any other way produce useful code for projects. The core developers are expected not to exclude anyone to submit code, though, as will be explored in later sections, some communities have created their own methods of filtering out potential unproductive members. Despite the apparent lack of incentives and the decentralized process,[4] OS software is not limited to simple games and programs but contends against companies with vastly larger budgets, in areas including operating systems, large online encyclopedias, and cloud computing services. OS somehow does more with less. Most projects are funded through charging money for the software, donations, and investments from commercial and OS firms. The phenomenon of OS, in general, it will be plain to see, is not a product of central planning but of individual actors using their varied and dispersed knowledge. In fact, OS software is often inhibited by government regulations and intervention, though some authors believe,

[3] A commit adds a change to some part of or the whole source code of the repository. Commits are usually easily rolled back if any issues arise.

[4] Another decentralized network involves peer-to-peer (P2P) sharing. The web of architecture which forms the communication network between diverse systems, allows for file transfer between peers, who act as both the client and the server. In this manner, a single peer does not store all the files nor transfer/share the entire workload, but the information is stored dispersed among users. Similar to the model of OS development, as will be explained later in this paper, failure on the end of a single peer does not disrupt the whole network.

as will be explored near the end of this paper, regulations and interventions promoting OS corresponds with an increase in general welfare and greater productivity.

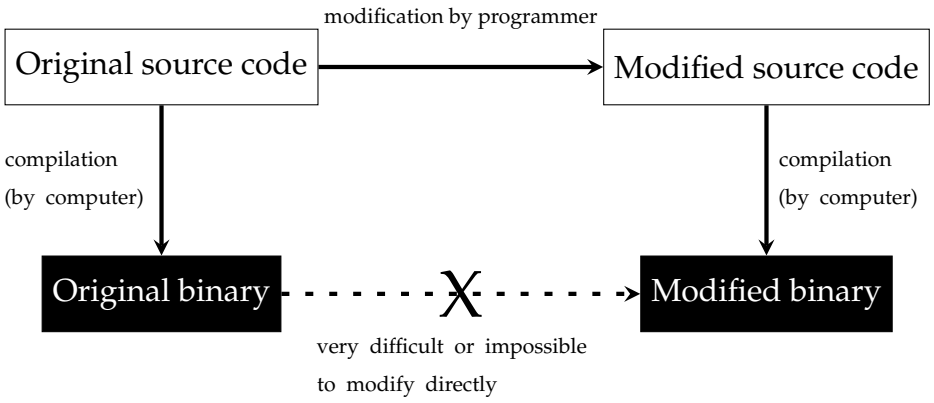
As a result, OS software is largely governed through private rules and private organization. I aim to build where other scholars also have alluded to, using Elinor Ostrom's useful work on common-pool resources (CPRs) and her Institutional Analysis and Development (IAD) framework. Though many great works have been done on the subject, in a short period considering the bareness of the literature at the turn of the millenia, in this thesis, I will explore the foundations of OS through three theoretical frameworks: (1) the framework of polycentricity (E. Ostrom, 1999), (2) the use of knowledge in organization (Hayek, 1945)—though other authors, relatedly, analyzed OS through granularity and modularity (Langlois & Garzarelli, 2008; Parnas, 1971)—and (3) entrepreneurial theory (Kirzner, 1997, 1999). For all parts, there will be some overlap though I will try my best to dissect and put back together the related nature of these parts, since the use and—potentially more importantly—the creation of information requires polycentricity. Curiously, OS operates on a large-scale coordination without the price mechanism, which runs counter to the arguments Mises (1920/2012) and many other economists made in the past and became to stronghold of economic theory.[5] Without prices, it is uncertain, then, how profit-seeking entrepreneurs in OS initiate economic change and if they are able to evaluate whether their efforts have beneficial or detrimental results. This will, accordingly, be explored in the entrepreneurial framework (framework 3). Lastly, the overarching question, which will be explored in the background throughout the paper, is, does the emergence of OS imply a "new form of industrial organization that capitalism creates" (Schumpeter & Swedberg, 1942/2003, p. 83)? Many scholars have argued the arrangement of OS constitutes a fundamental change in the market structure, while others are more critical about a fundamental change. For the rest of this introductory section, clarifications between open and closed source development will be provide and also the historical emergence of OS and the key individuals.

[5] Economists largely agree for a complex economic system, prices are necessary to direct the flow of resources and labor, even socialist economists Lange (1936) and A. P. Lerner (1934) concurred.

A. Open versus Closed

The access to source code is the fundamental difference between open and closed source development. Source code is the human-readable and modifiable recipe of a software program. To make source code run on any machine, it must first be compiled (i.e. translated) into machine-readable binary code, see Figure 1 for clarification. Machine code is simply a long string of zeroes and ones. Binary code is not easily readable, and therefore not easily modifiable, to humans—at least not at a rapid enough rate to create substantial programs. Access to source code, therefore, is important in order for OS to progress. Proprietary vendors, like Microsoft and Apple Inc., usually transfer their software as binary code to users. As a result, users must rely on the original vendor for any modifications or troubleshoot support. Users are individuals who interact and engage with software products but not directly with the code. For competitors or users to reverse engineer the programs would be impossible.

Figure 1.1 Source Code versus Binary Code



(Source: Schwarz and Takhteyev, 2010)

B. Historical Background

E. Raymond (1999/2002) was one of the first to outline the distinction between proprietary and OS software. Proprietary software, which Raymond compared to a cathedral, is centrally planned, tightly organized, and strictly monitored from beginning to end. OS, however, is more like a bazaar, where various and even conflicting goals and approaches are experimented with and implemented, but still results in a cohesive, functional, and stable system. By encouraging an open exchange of communication, ideas, and filesharing, collaboration

improves error correction and enables adaptation through various hardware platforms. This transparency and sharing are akin to the peer review process within publishing journals in academia. Benkler (2002) noted on the similarity between academics and OS developers because both are professional information producers, who autonomously select their projects and fields of inquiry, and then contribute their work into a common pool of knowledge where others are free to draw from. In other words, Raymond conceptualized software development in terms of top-down development versus bottom-up development. The analogy of a bazaar is often to describe the phenomenon of "spontaneous order,"[6] the idea which E. Raymond (1999/2002, p. 126) credited to Hayek.

One of the most prominent and complex OS projects is the operating system Linux;[7] with some people even believing Linux and OS are synonymous. In the late 1970s, the precursor to Linux Unix was first developed at AT&T, led by Ken Thompson and Dennis Ritchie. Originally, Unix was intended for commercial use with AT&T possessing licensing rights (i.e. proprietary software). Apple Inc. adopted Unix and incorporated their own elements, creating MacOS, which remained proprietary. Another Unix-like system, MINIX, was created by Professor Andrew S. Tanenbaum as a program for academic and educational purposes. After a few iterations, however, with the advancement of MINIX 3, the goal evolved into creating a fully functional operating system.

On the intellectual properties side, computer research and professor Richard Stallman, dissatisfied with the closed source development method both in the private sector and in academia, left MIT and founded the project GNU (GNU's Not Unix),[8] further discussed in the section Licensing. The groundwork has been laid for the development of Linux. In 1991 Finnish student Linus[9] Torvalds, age 21, was studying computer science at the University of Helsinki. Interested in the new capabilities of his new computer, particularly wanting to fully utilize the new Intel 80386 processor chip, he sought to write a program specifically

[6] Though the term "spontaneous order" is often associated with something good and natural, particularly among classical liberals and libertarians, there exist spontaneous orders which are detrimental such as negative beliefs systems, mob behavior (N. P. Martin & Storr, 2008), and traffic.

[7] See also other popular OS projects: the base version of Android (https://www.android.com/), the online encyclopedia Wikipedia (https://www.wikipedia.org/) the web browser Mozilla Firefox (https://www.mozilla.org/), the media player VLC (https://www.videolan.org/vlc/index.html), and the audio editor Audacity (https://www.audacityteam.org/).

[8] Recursive acronym.

[9] Linux is derived from Linus' Unix.

for this hardware. Torvalds had competition, though. During the same time, William and Lynne Jolitz at the University of California Berkeley also attempted to port Berkeley Software Distribution (BSD) Unix sources to the same chip. Finishing the precursory work, Linus asked the online community in a blog post with what features they wished to see on his version of MINIX.[10] Once the first version of the Linux kernel was released, early Linux developers ported GNU code to the compiler, adopting the use of the Linux kernel to the missing kernel of the GNU operating system.[11] Even to this date, some developers argue over the right terminology over the label of the operating system "Linux." Linux is technically the kernel used within the GNU operating system, which leads to some developers calling for the label "GNU/Linux." In contrast, BSD is both the kernel and operating system developed by the Jolitz's. There are more technical differences between the Linux and BSD systems, as well as differences in licensing between the two. The foremost being in Linux the release of source code must accompany any modification and redistribution, whereas in BSD modification and redistribution does not require the release of source code.

With Linux, we see firsthand one of the advantages of OS software: user involvement. The response from Linus' blog post transformed into a small project. A large community came together to work with Linus, which evolved into the development model for Linux. This development model has Torvalds making the final decisions of which code and features to add to the software, rejecting or accepting code submitted by volunteers. However, Torvalds yields his authority over to several trusted developers (designated as "lieutenants") who control different subsections of Linux.[12] The decision to accept or reject features and code rests largely on the discretion of these trusted developers.

Thousands of volunteers, programmers, testers, server maintainers, and other important roles have collaborated without (mostly) charging for their efforts. They have created networks of social and technological institutions to achieve their ends of creating complex and useful products. The emergence of OS software and its institutions seems to create a problem for the Coasean theory of the firm. Ronald Coase found it curious why firms should exist

[10] See https://www.cs.cmu.edu/ awb/linux.history.html

[11] The operating system is the software package which directly communicates with the hardware and the application. The kernel is the lowest level of the operating system, managing all of the system resources and translating user commands to machine language.

[12] Trust is typically thought to be "the extent to which a person in, and willing to act on the basis of, the words, actions, and another" (McAllister, 1995).

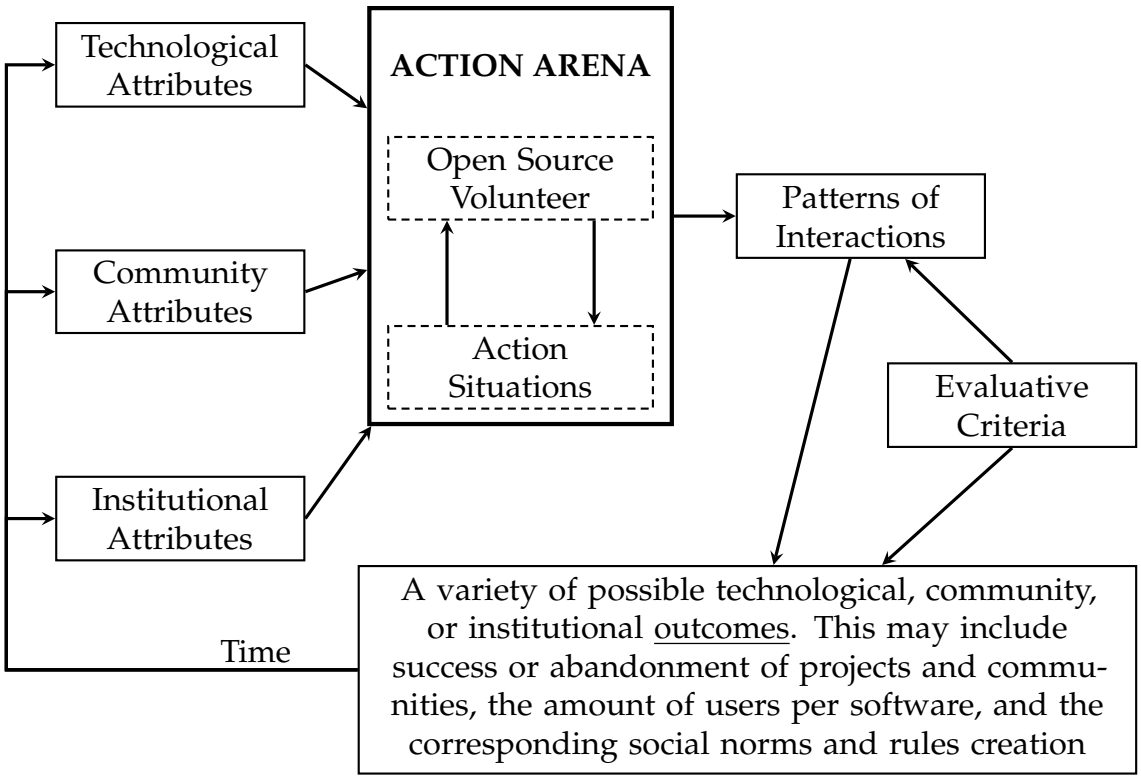
rather than individuals transacting on an open market. The informal and decentralized but structured and complex way OS develops would puzzle Coase even further. This puzzlement will be answered in the section Firm and Open Source Software Development. Classical and neoclassical economists by concerning themselves mostly with the efficient allocation of static resources typically ignore the relationship between individuals and their economic needs as a means to satisfy their subjective goals and ends. The rest of the paper provides a general overview of OS and potential explanations for its emergence.

II. Open Source Organization and Communities

When dealing with collective action problems, Elinor Ostrom's (1990) IAD framework encapsulates many ways to understand how these institutions, however varied and complex they may be, operate and change over time. Institutions in politics, OS, or other groups and associations are human-constructed constraints where individual choices take place. Indeed, the IAD framework can be used to "analyze dynamic situations where individuals develop new norms, new rules, and new physical technologies" (E. Ostrom & Hess, 2007a, p. 41-42). The entire OS community is made up of many smaller, nested communities, each of which may exist separately from each other or commingled. With rapidly changing goods, technologies, or communities, private governance and institutions allows for quicker adaptation, as will be explained further in this section. This section draws heavily from the work of Schweik and English (2012), who provides much more substantive IAD analysis—theoretical and empircal—of OS than possible in this thesis.

An initial conceptual problem when applying the IAD framework to OS is that knowledge commons have no biophysical characteristics. B. Frischmann, Madison, and Strandburg (2014, p. 21-22) noted, however, the process of software developement still contains many of the same properties of physical CPRs, including social interaction, man-made environment, laws, histories, practices, tradition, and social norms. As a result, however, Schweik and English (2012) adapted the IAD framework from Ostrom as a guide to OS (see Figure 2.1). In doing so, with this approach, Schweik highlighted the dynamic nature of OS development complete with a feedback mechanism. Using the terminology provided by Schweik and English (2012, p. 39-40), I will go over the terms used in figure 2.1. *Technological attributes* refers to

Figure 2.1 Modified IAD Framework for OS



(Source: Schweik and English, 2012, p. 39)

variables related to software production or the infrastructure necessary for the coordination of volunteers. This classification includes the type(s) of programming language(s) used, the modularity of the project (more on this later, including the concept of granularity), and the management infrastructure (e.g. Internet forum boards, messaging applications, servers to send and store code submissions, etc.). *Community Attributes* refers to a set of traits between the volunteers and the communities, such as the motivational reason for participating, the leadership and the level of trust regarding him or her, the amount of social capital and trust within the community, and the amount of volunteers and their interactions between each other. Group heterogeneity includes sociocultural, interest, and asset differences (Varughese & Ostrom, 2001), and as Varughese and Ostrom found, the difficulty which arises from these heterogeneous aspects can be overcome through good institutional designs. The financial situation and funding sources (e.g. nonprofit, private, and public foundations) are also included in this category. *Institutional Attributes* includes the management and governance of the community. This, of course, includes codified rules regarding coding standards (e.g. how to document code, formatting, patch procedures) as well as community and social norms.

In the middle of the figure, the *Action Arena*, is a conceptual unit which is the "focus of

analysis, prediction, and explanation of behavior and outcomes within fixed constraints" (E. Ostrom, Gardner, & Walker, 1994, p. 28). The *Open Source Volunteer* includes anyone participating directly with the OS development and the community (e.g. core code developers and reviewers, debuggers, independent submitters, funders, etc.). The *Open Source Volunteer* engages in *Action Situations*, where they "interact; exchange goods, services, and ideas; identify and solve problems; dominate one another; and/or fight" (E. Ostrom et al., 1994, p. 28). Over time, as a result of volunteers accumulating and retaining knowledge about one another and their skills, the more or less constants in their environment, and the needs of the users and community, *Patterns of Interactions* arise where the outcomes can be evaluated by every participant. In doing so, each participant can attempt to change each of the attributes or the action arena in order to facilitate what they surmise would be better patterns of interactions and outcomes.

The initial relevant factors Ostrom highlights for the IAD framework is: 1) biophysical charicteristics of the goods (such as the rate of production and return), 2) attributes of the community (e.g. proximity of interaction, age, motivations, etc), and 3) the existing rules in use. These factors constrain the action arena, which contain action situations (a platform or social space in which individuals of diverse preferences interact, exchange, solve problems, and form relationships) and actors. As such, it makes the most sense to identify specific participants and the roles they take. The IAD provides us the ability to see, as (Schweik & English, 2012, p. 81) puts it, "the glue that binds the technological and community attributes of OSS projects into a functioning whole."

Having provided a foundation towards the IAD approach, as a roadmap for this section, I will begin with outlining the collective action problem in relevance to OS (subsection A). Following that, applying Ostrom's IAD framework to OS as a CPR, I will examine how in OS rules came about, evolved, and changed the community. I will also provide a historical overview (subsection B) of the Free Software movement. In effect, how the movement started, the key individuals, and plans for the future. Continuing (subsection C), I will examine how OS attempts to move from proprietary licensing to a more open licensing regime, giving more rights to the public domain. Lastly (subsection D), the norms and rules in hacker culture and empirical examples of their communities (subsection E). In doing so, I hope to emphasize the

polycentric and dynamic nature of OS and its volunteers.

A. Open Source as Common-Pool Resource

An OS project typically begins with a founder or a core group of founders setting out to fix a small problem. Examples include: the origins of the World Wide Web (WWW) began when Tim Berners-Lee wanted to help high-energy physicists share their work; the origins of GNU General Public License (GPL) began when Richard Stallman wanted the source code to fix a problem with the photocopier; and the operating system Linux launched when Linus Torvalds sought to use and optimize a particular part of hardware. Eventually, as the amount and the scale of the problems the founder(s) wanted to solve grew, the founder(s) began to create a vision and plan for continual progress. Once code submissions from external volunteers come through, they essentially filter and implement whichever they believe will help their vision. Other highly motivated members become core committers. These core committers usually stay on with the project for a long duration, normally years. There is a much larger group of casual contributors motivated by other various factors. Again, some projects may simply be started for exploration and experimentation—there is not necessarily a utilitarian approach towards OS software development. Practical applications, like with theoretical mathematics or science, may arise outside and beyond the original intentions of the developers. A large majority, perhaps even all, OS founders would agree with the statement that proprietary software suffers from the *anticommons*. The tragedy of the anticommons is where a resource (material, knowledge, or otherwise) becomes underused not by natural scarcity, but because of restrictions caused by excessive private or government regulations and intellectual property rights.

As source code is deposited into the public domain, it appears to be an economic paradox as to why people continue to contribute to a CPR. There are three issues which initially arise. Firstly, unable to restrict and reap the benefits, investment in the project would be nonexistent. Secondly, without a hierarchical structure, there would be no ability to coordinate and effectively use resources. Lastly, there would be no incentive to maintain and update the project. However, in recent research started by E. Ostrom (1990, 2005), more analysis has gone into common-pool ownership. Earlier in the paper, OS was introduced

through more of a public goods theory lens. In this section, however, a CPR lens will be more explanatory on the nature of OS. Justification will now be provided in order to use a CPR approach in analyzing OS. Though CPRs typically involve goods or property which is limited and dwindles (i.e. subtractable), OS software does not. Copying and distributing finished software products have near zero costs.[13] Private plunder and depletion in public goods, as for the case of OS software, is impossible in this respect. Apesteguia and Maier-Rigaud (2006) differentiates public goods and CPRs in the same manner and while they found distinct strategic environments between the two, some Pareto solutions and Nash equilibriums remain the same in certain situations.

In relation to OS, a similar argument will be made, that is, under certain circumstances, OS as a public good is useful; under other circumstances, CPR is useful; and still yet for other circumstances, both may yield the same explanations. Kollock (1998) attempted to bring Ostrom into the realm of online communities, noting the challenges faced and solved in the physical realm could equally be incorporated into the virtual. The models—problems—Ostrom highlights for CPR (tragedy of the commons, the prisoner's dilemma, and the problem of collective action) also applies to OS communities. Likewise, (Williams & Hall, 2015) argued the physical spaces and equipment used within the space is a finite amount of resources, giving software development (Williams called them "hackerspaces") more of a CPR characteristic. Human creativity and time is also a limited resource. As Ostrom writes, "at the heart of each of these models is the free-rider problem" (E. Ostrom, 1990, p. 6). Finally, the polycentric[14] nature of OS communities, along with the previous justifications, lends itself towards similar analyses. OS communities, like the communities of individuals which Ostrom studied, rely on institutions not resembling the state nor the market to govern systems with reasonable success, though the existence of OS has been relatively shorter. Ostrom herself in the work *Understanding Knowledge as a Commons* (2007b) explored the "continual challenge

[13] There are still, of course, opportunity costs, which have indeed removed individuals from other pursuits, perhaps more productive pursuits, they could otherwise be doing. The time, labor, and human creativity used is then limited and subtractable.

[14] When speaking upon polycentricity and spontaneous orders, one must differentiate between the two, particularly in context of Polanyi (1951/1969) and E. Ostrom (1990, 1999), V. Ostrom and Ostrom (1999). As V. Ostrom correctly analyzed, Polanyi used the concepts of polycentricity, spontaneity, and competition interchangeably—the most in depth in the section Systems of Intellectual Order where he analyzed the competitive adjustments of science and Common law (Polanyi, 1951/1969, p. 162-165). Whereas E. Ostrom (1999) characterized spontaneity as a manifestation of "patterns of organizations within a polycentric system" (p. 59).

to identify the similarities between knowledge commons and traditional commons, such as forests or fisheries, all the while exploring the ways knowledge as a resource is fundamentally different from natural-resource commons" (p. 5). Having hopefully made the case of OS as a CPR, I will continue with the application of the IAD framework towards OS. As Ostrom wrote, the IAD is

> A general organizing tool that helps us develop a long-term research program not only for research on CPRs but also on other problems where individuals find themselves in repetitive situations affected by a combination of factors derived from a physical world, a cultural world, and a set of rules (E. Ostrom et al., 1994, p. 25).

More recently, many scholars borrowed and built upon the IAD framework to understand knowledge institutions (B. M. Frischmann, Madison, & Strandburg, 2014; Schweik & English, 2012). Therefore the IAD framework remains useful for explaining the emergence and governance of OS.

The pertinent problem in this section is how OS solves the problem of collective action and attempt to navigate (or completely avoid) the property rights system. Every individual displays varying willingness to initiate and return reciprocal behavior. Economists have long since recognized the role of reciprocity in society and markets since A. Smith (1776/1998, book 3), and more recently even in limited ultimatum and dictator games (Hoffman, Mccabe, & Smith, 2008).[15] While the primary goal of OS communities is to develop good software, a secondary function is to create a conducive atmosphere for reciprocity. Ostrom's work to answer the core question of "how potential cooperators signal one another and design institutions that reinforce rather than destroy conditional cooperation" will be useful when applied to OS (E. Ostrom, 2000, p. 138). She began with the idealized conception between the existence of two types of individuals: "conditional cooperators" and "willing punishers." For conditional cooperators, these are individuals who initiate cooperation given that they estimate a sufficient number of others will reciprocate and repeat these cooperative actions. The founders or core founders of OS projects fall into this categorization. Ostrom continued:

[15] Social norms for reciprocity is so strong and within human social behavior that the best condition to foster pure selfishness is in an environment of complete anonymity.

if the conditional cooperators reduce their contribution, they discourage other cooperators from contributing as well, leading to a downward cascade where eventually only the most determined conditional contributors continue to make positive contributions. If every co-ordination problem relies on all individual expecting the rest to contribute, cooperation would have never existed. Therefore, it seems to follow, some individuals through their own propensity and reasons, initiate and continue contributions even if no one will reciprocate. These individuals inspire and motivate others, much like community and religious leaders Chamlee-Wright (2010) described in her work. The question is which institutional environment allows these individuals to guide growth in the most productive manner.

Figure 2.2 Design Principles

1. Clearly defined boundaries
2. Create rules and distribution adapted to local conditions
3. Ensure those obtaining and using the resources can modify the rules
4. Ensure accountability for monitors of the rules
5. Match punishment proportional to the type of violation
6. Allow cheap and easy access to conflict resolution
7. Maintain the sovereignty and self determination of the members of the community
8. For larger CPRs, allow smaller nested common-pool enterprises to form

(Source: E. Ostrom, 1990, p. 90)

Most influentially, she developed eight "design principles" widely application to stabilize local CPR management (Figure 2.2). Although the software itself in irreducible and provided for at near zero cost, the creativity, human development, and time involved in OS software is not. In this sense, OS development includes subtractable components. As such, in an attempt to self-govern, self-restrict, and self-promote software submissions, review, and maintenance, OS must discover, incorporate, and modify formal and informal rules. Remarkably, most OS software exhibit many if not all of the eight design principles formulated by Ostrom pertaining to CPRs. The next few sections will highlight how OS expresses these principles. Though, as will be later explained, there is a spectrum between OS and proprietary firms, and at each end, there are clearly defined boundaries of what constitutes OS. The software created, though most of it is in the public domain, has different licensing systems which ensure flexibility for users and developers. The eight principles will be explored throughout

the rest of this section in Free Software Movement, Licensing, and Hacker Community and Culture.

While early OS projects started in firms and academic settings, today many small projects are founded by an individual or a small group with an abstract concept of what they wish to accomplish. Often they start a project to solve a small problem or for intellectual exploration and pleasure. New code or revisions (known as "patches") are either submitted through popular online managers like the websites Github[16] and SourceForge[17] and other online repositories. These patches can be as simple as a one-line bug fix or 5000 lines of a new feature. Proper submission procedure include a description of the problem or feature to fix or implement. Additionally, the code should be well documented—there should be written texts or illustrations on how the code operates and its detailed uses. Though these formal rules are not followed in some projects, this is a problem of time prioritization between working on other avenues of the project and providing adequate documentation. OS projects with more contributors and better popularity will no doubt attract volunteers to fill in these roles. There has been no attempt to determine whether the success of certain projects is first due to proper documentation if it is the other way around, that success leads to propert documentation, though it seems more reasonable to assume creating a better product should first attract more users, which leads to better documentation.

In the case of Linux, layers of different reviewers and maintainers, each with separate understandings of each subsection, will look at the code and judge whether the problem is sufficient and whether the code provided will solve it. In effect, there are still guidelines and conventions to follow despite the absence of a central authority. In evaluating the received code, there will be a balance between performance, memory consumption, stack footprint, and binary size. In other words, trade-offs will be evaluated between the cost of running the new feature will be justified by its benefits. Certain software are designed specifically for low-end machines with limited hardware (e.g. memory, storage, CPU clock speeds) and developers will keep this in mind when evaluating additional features. Though most OS communities is open access, boundaries are set against those who submit viruses or other

[16] Github provides a web-hosting service which is used to distribute software but also includes features such as bug tracking, feature requests, task management, and access to various forms of information and documentation about the project.

[17] https://sourceforge.net/

malicious lines of code. Harmful submissions will quickly be found by the higher skilled developers in the project and by the diverse knowledge provided through the nature of the open community—exemplified by the quote: "given enough eyeballs, all bugs are shallow" (E. Raymond, 1999/2002). Moreover, the reputation associated with other identifiers like emails, usernames, or IP addresses will be remembered. Given the difficulty of some OS communities to join and contribute, this, for the most part, is an effective filter against harmful behavior. As discussed earlier, some communities, while technically open to all, will only allow members who consent and abide to the community constitutions and rules, creating a sort of private club. These rules reduce monitoring costs.

Each project release will have a uniquely identifiable version number. Though there is no formal convention for every OS project, these numbers are not arbitrarily assigned. Though version numbering scheme is up to personal preference, individual or community, two schemes are commonly used. For public applications, *Major.Minor.Patch* is used. This is the most popular and most adopted scheme. Every change on major, minor, or patch number must incrementally increase after each revision. A change in the major number can only be made if the application program interface (API) is not backwards compatible with previous releases. A minor number is for backwards compatible feature additions. Patch number is for backwards compatible bug fixes. Additional numbering regarding pre-release and build metadata can, logically, extend the *Major.Minor.Patch* format to *Major.Minor.Patch.Pre-release.Metadata*.[18] For in house applications, *Year.Monthy.Day.Build* is used. This allows internal information technology employees to receive more information on the project version and how to fix software problems in more situations.

There are around 304 active Linux distributions[19]—"distros" for short and also more uncommonly called "flavors"—which all have separate development and release dates, although many programmers are part of different OS communities and participant in various projects. This makes sense as Mises wrote, "individuals do not simply belong to one community or group but take part in a multitude of human affairs" (Mises, 1957/2007, p. 257). Any club, group, community, and other voluntary memberships must obligate a social philosophy of

[18] See https://semver.org/ for additional rules and information for specific situations.

[19] According to the site https://www.distrowatch.com/, which provides a collection of all available distros and ranks them by popularity, with news on updates, the popularity, tutorials, and other resources.

individualism and polycentricity. Of course, this is not limited simply to OS communities but includes clubs, religious communities, trade associations, homeowner's associations, and fraternal organizations. These groups may overlap in some dimensions or remain strictly divergent. The organizational structure, formal and informal rules, and individuals remain different for every OS project and each community—this solidifies Ostrom's aversion to a one-size-fits-all approach. This polycentric approach, though E. Ostrom (1999) argued in terms of governing institutions, equally applies to OS communities, and allows developers and users to find more opportunities to correct organizational and social errors.

Each distro is designed with different functions and users in mind. Operating systems ranges from graphically user-friendly interfaces (designed similarly to Windows and MacOS to help users transition) to others with text-based interfaces to yet others with no pre-installed software other than the bare number of packages to start running. Certain distros are designed for specific tasks, including network monitoring, server administration, or office and academic work. Individuals may build and compile their own private distro and implement their own code and features. When certain individuals or groups become discontent with the progress and decide to take the existing project and incorporate their own updates in parallel to the original—this is known as "forking." Many projects that were forks can and have been developed into independent and progressed into entirely different projects than the original. Furthermore, there are different variations of platform competition and project and community divergence. The definition of forking remains contended. Distinguishing between forking vs. fragmentation vs. code reuse remains difficult as it would seem at first glance the distinction is merely between degree rather than kind. Building on the definitions provided by Nyman and Mikkonen (2011), fragmentation occurs when all parties would like to adopt a common standard, but cannot agree on what it should be. For instance, when the suppliers of the Digital Video Disc (DVD) format split into the incompatible Blu-ray and HD-DVD formats. Code-reuse from one project to another may be considered a fork by some, depending on how much code is borrowed. Regardless of differences over definitions, most OS projects want to strike a balance between the right to fork and the right to standardize.

Control and transfer of control for OS projects occurs in three other ways other than forking: (1) to found a project, (2) from transfer by a previous person in control, and (3)

volunteer to take over a dying project. Therefore, larger OS projects are able to split into other smaller, nested OS projects. Although the decision to fork is heavily stigmatized—it rarely happens—this feature is an "exit strategy" (Hirschman, 1970; Kurrild-Klitgaard, 2010) and incorporates boundaries that includes exit rules (E. Ostrom, 2005, p. 198-200), which limits the extent of sustaining and perpetuating destructive social interactions.[20] Forking, thus, facilitates the conditions for a polycentric order within OS development and allows the self-governance of each project and community. There are many reasons to fork. They may be planned, temporary divergences to test new ideas and features, with intentions to conjoin later. Nyman and Mikkonen (2011) analyzed 381 fork projects on SourceForge and found the main fork motivation for a majority (72 percent) of the projects was related to the content and technical matters of the project. Content-related improvements not only include the self-explanatory features, but may also involve advertisements, blogs, and podcasts about the project. Technical modifications can be divided into two subgroups: porting and improvements. Porting software means changing the original code to fit another operating system, hardware, plugin, or to make compatible with other hardware or software specifications. Improving the original code is a rarer reason for forking since the original project can easily incorporate such code—indeed, this is main the reason for OS development. Though there are other channels of influence, the primary mechanisms, then, are to submit patches or to exit (by forking or leaving the community). Different projects, depending on how strongly they value conformance with a norm, as a result, have different implementation strategies. In BSD derivatives, a concentric circle of hierarchical members makes the decisions, with core developers making major decisions and contributors submitting modifications for evaluation; for Linux Torvalds is a benevolent dictator (with essentially a parliamentary system of trusted developers; for Perl there is a rotating dictatorship, where developers take turns making

[20] An earlier notion of this idea, "foot voting," while not explicitly termed by Tiebout (1956), was conceptualized through his model of consumer-voters moving to the locality that best suited their set of public goods preferences.

decisions for the project (Weber, 2004, p. 92);[21] other projects, such as the Apache web-server create a voting committee (Fielding, 1999), where a system of voting via email based on a minimum amount of consensus is used. Regardless of common preconceptions involving each system (the epistocratic, dictatorial, monarchic, and democratic style of governance) and though these governance structures only involve OS communities, not in the political realm where these governance structures bear greater power and influence, OS offers continual exploration and re-exploration on what works best for each unique community.

Different systems also create different conflict and resolution outcomes. Conflicts that do not escalate to the point of chaos may result in new understandings. As long as the decision-making process is agreed to be legitimate *ex-ante* and if there is some method for which these rules may be changed, building and maintaining OS software will be more robust and sustainable. Communities and projects that contain overly rigid and fixed rules may fail when an unexpected problem arises. Experimentation for which system decides how much feedback and core control is necessary for every project and the corresponding OS communities. With larger communities, the decision-making process may become unwieldy since although this is dependent on how the community is organized, there could exist smaller subgroups that form within the larger community, creating a more complex social terrain for newcomers to navigate. The amount and span of control must be contextualized and flexible. In changing one of the rules, however, more often than not, will affect the remaining sets of rules, since these rules would have evolved together. It is up to experimentation, on the margins of each and every rule, that the community can change effectively. Without this cautious approach, collapse of the community is more likely. Moreover, with more individuals, forming any sort of consensus to incorporate certain features or changes will increase in difficulty, with increasing organizational and transaction costs. For collective goods, including OS software, the larger the group, the more difficult it will be to optimize resources and to underproduce those goods. There is, then, a tradeoff between the size

[21] "Larry Wall, the originator of the programming language Perl, in the mid-1990s developed a different version of a delegated decision-making structure. There is an inner circle of Perl developers, most of whom took an informal leadership role for a piece of the code during the transition from Perl version 4 to Perl version 5. Wall would pass to another developer the leadership baton, that person would work on a particular problem and then release a new version of the code, and then pass the baton back to Wall. This process developed into what Wall called "the pumpkin-holder system." The pumpkin holder acts as chief integrator, controlling how code is added to the main Perl source. In a kind of rotating dictatorship pattern, the pumpkin gets passed from one developer to another within the inner circle" (Weber, 2004, p. 92).

of communities and the perceptible difference in contribution (degree of granularity and modularity).[22] The more members, the less work each member will have to do in order to meaningfully contribute but there is also an increase in complexity and costs for coordination and social cooperation. In projects where the OS community is both large and requires a high proportion, or even unanimous, agreement to select patches or features will result in slower release dates, or even collapse of the project. Applying Olson's (1965/1971) group analysis to OS, different of individuals will favor different features, and individuals with the same—or similar enough—preferences will most likely form small subgroups in order to bargain or convince others to implement those preferences, which will increase the cost of decision-making.

B. Free Software Movement

The communities formed around Linux and other OS software projects have inspired a movement for free and open source software (FOSS).[23] The word "free" in this context is not about pricing but pertains to liberty.[24] Others, however, argue all information, including software, should not have to be paid for (Levy, 1984/2010, p. 46). There has been some debate over the definitions and applications between the terms "free software" and "open source software." For some, all code should be open, for others all code should not necessarily be open but should be zero cost, and for still some others all code should be both open and zero cost. There are further nuances between each of these distinctions regarding the types of code and projects. For the most part, though, many OS participants believe those who purchase programs should have the freedom to study, distribute, and modify the code within

[22] There are many more variables researchers found to be conducive or detrimental towards collective action: "the type of production and allocation functions; the predictability of resource flows; the relative scarcity of the good; the size of the group involved; the heterogeneity of the group; the dependence of the group on the good; common understanding of the group; the size of the total collective benefit; the marginal contribution by one person to the collective good; the size of the temptation to free-ride; the loss to cooperators when others do not cooperate; having a choice of participating or not; the presence of leadership; past experience and level of social capital; the autonomy to make binding rules; and a wide diversity of rules that are used to change the structure of the situation" (E. Ostrom, 2000, p. 148).

[23] This movement runs in parallel with other "open" movements, such as open data, open content, open innovation, open data, and open government, just to name a few.

[24] Stallman often quips, the concept of free to the freedom of "free speech," not as in "free beer." Sometimes also referred to as *libre* (at liberty) software, as opposed to *gratis* (free of charge) software. To read more, see https://www.gnu.org/philosophy/free-sw.en.html

that program.[25] In this sense, most OS software and free software satisfies both definitions, but not always (Carver, 2005). The Freedom Software Definition (FSD) is comprised of the four freedoms:

> **Freedom 0** The freedom to run the program as you wish, for any purpose.
> **Freedom 1** The freedom to study how the program works, and change it so it does your computing as you wish. Access to the source code is a precondition for this.
> **Freedom 2** The freedom to redistribute copies so you can help others.
> **Freedom 3** The freedom to distribute copies of your modified versions to others. By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.[26]

These freedoms include the right to copy and change, even sell, copies of the software. The Open Source Definition (OSD) is more detailed, listing ten criteria. The first two criteria being:

> 1. The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.
> 2. The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost, preferably downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.[27]

Software licenses under GPL, like Linux, is satisfies both definitions. For the most part, the OS movement is more open into entering commercial streams while FSD is taking a stricter position regarding commercialization, often confronting proprietary firms, like historically with Microsoft.

Indeed, both the FSD and OSD runs adjacent to the thoughts of Thomas Jefferson:

---

[25] A similar problem arose during the advent of music stored in compact discs (CDs). Consumers would buy CDs, copy (rip) the music, and store the information on their computers or external drives for personal use or sharing. Though the music was owned by the musician and the producing label, enforcement of this intellectual property proved nearly impossible. As a result, successful music platforms today shifted focus to streaming services, though a vibrant community of those who enjoy music through vinyl exists.

[26] https://www.gnu.org/philosophy/free-sw.en.html

[27] https://opensource.org/osd-annotated

> That ideas should freely spread from one to another over the globe, for the moral and mutual instruction of man, and improvement of his condition, seems to have been peculiarly and benevolently designed by nature, when she made them, like fire, expansible over all space, without lessening their density in any point, and like the air in which we breathe, move, and have our physical being, incapable of confinement or exclusive appropriation. Inventions then cannot, in nature, be a subject of property (Jefferson, 1813/1905).

With copyright laws being extended to much as 95 years, the FOSS movement creates a counterbalance against excessive monopolistic control over information and knowledge (Dorman, 2002). Along with Torvalds, Bruce Perens, Tim O'Reilly, Richard Stallman, and many others from academia, corporate, or media backgrounds joined in supporting the free software movement by creating OS business model firms, legal advice for patent and licensing infringement, and in the intellectual sphere. For them, intellectual monopolies produce more harm than advantage to the society and by negating such protectionist policies would produce fruitful, new, and useful software.

C. Licensing

Though the main goal of OS is software availability to everyone without restrictions, there is a spectrum of rights for intellectual property, with some licenses retaining more or less of the rights to modify, distribute, or sell; see Figure 2.3. Patents, on the whole, argued by Kapczynski and Syed (2013), consists of a "continuum of excludability," where existing social norms and institutions provide "acceptable" or "reasonable" enforcement. These norms and institutions are subject to change. As such, there are associated transaction costs to police and enforce agreements (Williamson, 1985)—even pertaining to OS. Licenses (occupational, patent, academic, and etc.) is an official permit or permission to do, use, or own something—which necessarily restricts the amount of said thing. Government mandated licenses may increase the risk of corruption, restrict the supply of goods and services (create shortages) which for some supplies some scholars argue may be desirable, and limit the amount of competition from newcomers (thereby leading to lower quality). On the other hand, historically, licensure was intended to create a competent minimum barrier of entry. Another reason may be

to incentive businesses for research and innovation which may not be easily internalized. Exclusion costs, internal governance costs, and implementing punishment for violators all make private property costlier (R. N. Johnson & Libecap, 1982). Overly restrictive control of entry will inhibit initial and long-term participation in development. OS software, for the most part however, requires little of the benefits from licensure in order to succeed. In fact, the opposite is true; many OS developers and volunteers find licensure obstructive to their goals and innovative capacity. OS attempts to circumvent these issues under non-protective and protective FOSS licensing, where individuals are free to use, display, copy, and modify. They can redistribute the software only if under the same license. Licensing models of this type require any derivative work to reciprocate all rights. This is known as "copyleft" licensing. See Figure 2.3 for a generalization on the types of software licences.

Figure 2.3. Classification of Software Licenses

Public Domain	Non-Permissive FOSS License	Permissive FOSS License	Proprietary License	Trade Secret
All rights relinquished	⟸More rights granted		More rights ⟹ retained	All rights retained

(Source: Mark Webbink[28])

Richard Stallman left MIT, previously working on artificial intelligence development, to develop "free" software. Stallman found it ethically imperative to achieve this goal. It was fair, he believed, not only to have access to executable and binary code but also the source code as well, so that others may learn and improve upon it. To start his vision, he founded the Free Software Foundation (FSF).[29] He would create a new model of licensing to implement his vision: the GNU General Public License (GPL). The GPL allows developers to use existing copyright laws to protect their work and, at the same time, allows license holders the freedoms of OS (i.e. copy, use, modify, and distribute[30]), provided they provide recipients with equal rights of use—that is, GPL is the very first copyleft licensing model. As of writing this thesis, the GPL is the most widely used OS license. Under the GPL license,

[28] Mark Webbink is a visiting professor at New York Law School, a senior lecturing fellow at Duke University School of Law, and a board member of the Software Freedom Law Center. He has written and researched prolifically on OS and intellectual property issues.

[29] A 501(c)(3) non-profit organization, see https://www.fsf.org/

[30] The freedom to distribute includes either for free of charge or for a fee.

developers can know whatever software progeny which comes after, no matter how far down the line, will remain nonexclusive. Richard Stallman along with Eben Moglen navigated the issues of the licensing scope, dynamic linking, and Linux kernel modules. Tsai (2008) analyzed the evolution of the GPL from its initial version to its current third revision (GPLv3). GPLv3 incorporates new language and terms to clarify previous ambiguities, though Tsai (2008) argued new language inevitably introduces new ambiguities, and GPLv3 remains problematic when dealing with proprietary and more restrictive software patents. Still, the attempt at producing an atmosphere of freedom is vital for continual access and growth of OS.[31] Other licenses allow for different actions and freedoms. The BSD license, for example, allows for many of the same freedoms as GPL, though BSD does not mandate follow-on works to be distributed free of further restrictions. Therefore the contributor of original code under BSD retains the copyright on the contribution and is still able to release that part of the code under proprietary license terms. See Table 2.1 for more information on the differences between the most common software licenses.[32] The success of OS is not entirely due to the legal structure, rather, GPL and other OS licenses works are because there are communities which accept their use. As O'Mahony (2003, p. 1189) wrote, "the GPL strength stems not necessarily from its legality, but from the public collective opinion of community members." Sen, Subramaniam, and Nelson (2008) argued intrinsic motivations (creative pleasure and self-challenge), extrinsic motivations (status and economic opportunity), and developer attitudes (on users' rights, OS redistribution, and social obligation) all affect which license(s) the developer will choose for his or her software.[33]

Another option for licensing are dual licenses. Analyzed by Deek and McHugh (2007, p. 61-68), Sendmail and MySQL AB provides a viable business model. In both models, the companies obtain an OS license and a proprietary one. This allows the companies to continue to operate as an open product while at the same time maintaining goodwill with the other

[31] As Mill (1859/2001, p. 60) wrote, "Persons of genius, it is true, are, and are always likely to be, a small minority; but in order to have them, it is necessary to preserve the soil in which they grow. Genius can only breathe freely in an atmosphere of freedom."

[32] Additionally, see https://tldrlegal.com/ for summaries on other licenses, end user license agreements (EULAs), and terms of services (ToS).

[33] Sen et al. (2008) simplified the spectrum to three categorizations: strong copyleft, weak copyleft, and non-copyleft. From their study, while intrinsic motivations influenced licensure heavily, developers did not always choose the least amount of restrictions on their software. Trade-offs between the OS philosophy and the impact on project activity and success, it would seem, creates differences in licensing preferences.

companies using their products. The commercial version of the software competes with the OS version, resulting in increased pressure for developers to incorporate features differentiating the two. The benefits of OS (e.g. bug spotting, pooling diverse knowledge, potential for increased innovation, and etc.) can be incorporated into the proprietary version, while at the same time including more specific features the commercial companies desire. Independent voluntary contributes can submit patches, though the company's own development team will reimplement them if deemed acceptable.

Table 2.1 The Most Popular OS Licenses and Their Rights

License	Link	Distribute & Modify	Patent	Private Use	Sublicense	Trademark
Apache[34]	Permissive	Permissive	Yes	Yes	Permissive	No
BSD[35]	Permissive	Permissive	Manually	Yes	Permissive	No
GNU General Public License[36]	GPLv3 only	Copyleft	Yes	Yes	Copyleft	Yes
GNU Lesser General Public License[37]	Restricted	Copyleft	Yes	Yes	Copyleft	Yes
MIT / X11[38]	Permissive	Permissive	Manually	Yes	Permissive	Manually
Open Software License[39]	Permissive	Copyleft	Yes	Yes	Copyleft	

Going over the terms used in Table 2.1, *linking* is the use of an external reference; for example, using a library or module under a different license. *Distribution* is sharing the code to third parties. *Modification* is the changing of original code by the licensee. *Patent* grant protects licensees from patent claims made by other license holders. *Private use* is whether the license allows modification to the code to be shared in a community or internal use in a corporation. *Sublicensing* is whether the license allows for the modified code to be under a different license (i.e. copyright) or must retain the same rights under which it was provided.

[34] http://www.apache.org/licenses/LICENSE-2.0
[35] https://opensource.org/licenses/BSD-3-Clause
[36] https://www.gnu.org/licenses/gpl.html
[37] https://www.gnu.org/licenses/lgpl.html
[38] https://opensource.org/licenses/MIT
[39] http://rosenlaw.com/OSL3.0-explained.htm

Trademark grant allows the use of trademarks under licensed code or modification of code. The private initiative of licensing, at least in the case of OS, though imperfect, has far more flexibility and more easily allows for future adaptation than the attempts of the Federal Communications Commissions (FCC) to license radio and other broadcast content (Baumol & Robyn, 2006). Open licensure allows for adaptive arrangements rather than vested interests to protect property rights.

Legal cases involving OS licensing has been relatively few compared to proprietary licenses. One of the largest legal cases pertaining to OS was filed 2003 and is still ongoing between Santa Cruz Operation (SCO) Group, Inc. and International Business Machines (IBM) Corporation. AT&T Technologies, Inc. was the original owner and developer of Unix. Eventually, the SCO purchased Unix, becoming the first UNIX company. Since the 1980's, IBM developed and marketed its own version of Unix. In 2003, SCO filed a $1 billion lawsuit against IBM in Utah, citing four actions: misappropriation of trade secrets, unfair competition, interference with contract, and breach of contract. SCO argued IBM's version of Linux contained unauthorized derivatives of UNIX source code, which SCO owned legal rights to. IBM filed a countersuit, claiming SCO violated some of IBM's copyrights and patents. As Goettsch (2003) rightly predicted, the case has not been easily settled because of both players having enough funding to sustain long-term legal battle. Even though SCO filed for bankruptcy in 2007, the case was reopened under a new judge in 2013. The judge granted mostly in IBM's favor in 2014 and dismissing the rest of SCO's case in 2016 with prejudice. Later in March 2016, SCO filed for an appeal. The outcome largely affects OS and future progress in this area. Furthermore, not limiting their legal action to IBM, SCO issued warnings to over 1,500 other companies notifying them their use of Linux within their business was a violation of their property rights, threatening litigation under continued use. Three other major cases were filed from SCO against AutoZone,[40] DaimlerChrysler,[41] and Novell; while Red Hat filed a case against SCO. Noticing the cases SCO had against other businesses using Linux, Red Hat feared they would be next, thereby asking SCO for a permanent injunction and a number of declaratory statements that Red Hat has not violated

[40] Settled a with confidentiality agreement in 2009.

[41] In 2004, SCO filed suit against DaimlerChrysler for violating their UNIX user agreement. The parties agreed to a stipulated dismissal order at the end of that year.

SCO's copyrights.

OS and open license pose a novel problem to the judicial system. Indeed, B. K. Thomas (2010) contended FOSS is radical, promoting users into active participants in development and achieving greater democratization than in the political sphere. Under GPL, not only can developers charge for their software, it is their guaranteed right. When copyright in GPL is infringed, many of the cases are sorted through private means and settlements, with the Free Software Foundation providing *pro bono* general counseling and defense. Additionally, one of the more prominent private methods of enforcement comes from Harald Welte, founder of gpl-violations.org, which is a non-for-profit project to help track down GPL violations. The goals of the project, from the website, is to raise awareness of possible infringement of free software, give users the ability to report infractions, assist in copyright holders to take action on any parties infringing, and to distribute information on how to comply with the GPL license. The Software Freedom Law Center, founded by Eben Moglen,[42] also provides *pro bono* licensing defense and litigation support. OS legal troubles, remaining consistent, requires OS solutions.

Founded by Lawrence Lessig in 2001, the non-profit organization Creative Commons is used by content-sharing platforms like Wikipedia and Youtube. The website[43] helps creators choose the right license to share their work. If the creator wants any adaptation of their work to be shareable, they are able license their work in such a manner. If they want to allow or deny commercial reuse, that is possible as well. However, additional legal or technical restrictions not explicitly voiced in the license cannot be implemented. Use and remix, along with the correct attribution of the original work, allows photos, videos, writings, music, and other creative content to be open for public use.

Other than formal licensure, O'Mahony (2003) examined six other tactics to protect against proprietary appropriation of OS code and software: (1) encourage compliance with licensing terms through normative and legal sanctions; (2) incorporate to hold assets and protect individual contributors from liability; (3) transfer individual property rights to collectively managed non-profit organizations; (4) trademark with brands and logo designed to represent

[42] Eben Moglen is a professor of law and legal history at Columbia University and has been involved in intellectual property patenting and has been heavily influential in drafting OS licenses.

[43] https://creativecommons.org/choose/

their work; (5) assign trademarks to a foundation; and (6) actively protect the project's brand.

D. Hacker Culture and Communities

Cultural norms heavily guide the development of OS in an informal manner. By exploring OS culture, explanations regarding how social behaviors arise, the recognition mechanism to reputation, and how OS protects each other from legal ramifications will be explored. OS culture governs, for the most part, the ideology of the members and the behaviors of the members. During the rise of the computer age, "hacker culture" came to challenge what it meant to be "masculine," redefining masculinity as mastery of technology, independence, and confronting adult authority (D. Thomas, 2002). The term "hacker" has been often mistaken and misused in the mainstream. Rather than being programmers or developers who commit cybercrimes or break security systems on machines, the term hacker originally pointed to a connected culture—or group of cultures—which has existed since the 1950s. Hackers were hobbyist programmers who built software and gave away programs—one of the most important products was the World Wide Web—and they sought to preserve *libre*, or pertaining to liberty, in programming culture based on hacker ethics. To be called hacker in the OS community is a badge of honor. "Crackers," those who *do* commit cybercrimes or break security on machines, recognized hackers were much more skilled and appropriated the term for themselves to sound more capable ("Eric Raymond on Hacking, Open Source, and the Cathedral and the Bazaar", 2009).[44] The term "hacker" underwent a shift in negative connotative meaning during the 1980s and 90s. Hackers came about, and later OS communities, due to the inadequate nature of firms for fulfilling some values individuals desired. The absence of openness and sense of meaning led to individuals who are more inclined towards these values to start projects which sacrifice some profitability in order to find meaning—for some projects, the sacrifice of all monetary returns. Other contributors, while also holding these values but are unable to live an ascetic or more minimal lifestyle, contribute to OS while still holding jobs through firms.

Sauer (2007) expanded on why individuals and hackers (although he did not specifically study hackers, though their motivations remain similar to other programmers) develop OS software. They can build on existing OS code and develop software, which yields higher

[44] Similar to contemporary American liberals misappropriating the label "liberal." (Hayek, 1994, foreword).

future returns. In many instances, solutions created for a single problem can be more generally applied to a larger class of users, particularly solutions involving algorithms and methods (Haefliger, von Krogh, & Spaeth, 2008). In other words, the software is maintained through reciprocal giving. Additionally, human psychological behavior governs individuals in publicly releasing their code. Another reason highlighted by Sauer is the signaling role of OS development. Code submission builds reputation about the skill of the hacker but also signals to the other members of the community their beliefs and commitment to the openness of knowledge. This is particularly useful for inexperienced programmers, as it builds rapport for future employers and creates a stronger sense of reciprocity within the OS community. Changes in hacker culture, therefore, were endogenous OS developers are not subject to similar high-pressure corporate deadlines which, along with the low cost of submission, has allowed Linux and others OS projects to continue to grow. Projects with more centralized structures and the increase in associated transaction costs, such as the Free Software Foundation, have higher costs of submission and therefore growth slower than comparatively decentralized structure, like Linux.

New members and contributors who may eventually become hackers are created through word of mouth and the general diffusion of information. Some communities though, like their software, is open to everyone, like membership for the GPL. Other projects, on the flip side, are not—and for certain communities, membership is extremely restrictive. Steep learning curve involving learning the required programming language(s), correct submission procedures, other technical knowledge, and social convention are but a few barriers which limit potential newcomers. By self-teaching oneself the norms, this signals not only hard work, self-guidance, and dedication but also a similar mindset and the same general ideology as the rest of the community (Stewart & Gosain, 2006).[45] First though, Krogh, Spaeth, and Lakhani (2003) showed individuals observed and particpated in public project forums before becoming active members of the OS community. Newcomers are expected to join mailing lists,

[45] For a similar reason, the open collaboration of Wikipedia restricts the editing and revising of the articles to only those who "understand the norms, socializes himself or herself, dodges the impersonal wall of self-automated rejection, and still wants to voluntarily contribute his or her time and energy can edit" (Halfaker, Geiger, Morgan, & Riedl, 2012). In overcoming this barrier, newcomers signal commitment of high quality. Open is never truly open participation in this sense. Social norms and informal rules will always eventually form to create order and direction. From a distance, it may seem barriers are created in order to keep out newcomers, and in a sense this is true, but it also facilitates sharing and a cooperative atmosphere. As Strahilevitz (2003) argued, "charismatic" technologies create more communities seem more cooperative than it really is.

discussion boards, and view publically available information (including previous discussions; a behavior known as "lurking"). In some projects, budding members who have gone through the prerequisites are then brought into mentorship programs. Mentorships are rare though, partly due to the physical distance and the high turnover rates of constributors (Park & Jensen, 2009) and Park and Jensen (2009) concluded online visualization tools would better help newcomers find information more quickly and effectively. The Linux Foundation additionally offers certification programs in systems administration, cloud computing, open stacking development, and more.[46] Once newcomers become familiar with the norms, values, and technical skills of the community, to progress further, they must carve out an identity for themselves and how their particular set of skills may contribute to the project (Steinmacher, Conte, Gerosa, & Redmiles, 2015). Rules that these volunteers begin to follow, however—and this bleeds into Hayek's work on the dispersal of knowledge—may not be explicitly articulated,[47] but nonetheless followed in combination or better perhaps, in conflict with other impulses which, depending on the strength, will determine whether the rules or impulses will prevail. Rules, like laws as Hayek (1973/2013) argued, are initially discovered—not created—from codes of conduct already existing in the community. After codification, developers may, in turn, purposefully add, modify, and remove rules, though the way these rules are interpreted and subsequently applied is ultimately up to the individuals in these communities. Rules are not self-enforcing and any rule runs in danger of yielding unintended consequences. Due to the elegant and creative nature of code creation, submissions are accepted on a meritocratic basis, with core developers examining the effective use of resources and conciseness of the code. The ability to make changes to Apache, for example, is comprised of people who have been chosen because of proven ability and past contributions (Kogut, 2001, p. 254). Of course, all of this makes sense. Sifting through and understanding frivolous and harmful submissions, in a large enough project, would be laborious and time consuming. Norms and rules are created in order to mitigate *ex-ante* these problems.

Emulation drives much of human behavior. As Mises (1957/2007, p. 294) explained,

[46] https://training.linuxfoundation.org/certification/

[47] Hayek (1973/2013) wrote, "It has made clear that individuals had learned to observe (and enforce) rules of conduct long before such rules could be expressed in words" (p. 71) and "So long as language is not sufficiently developed to express general rules there is no other way in which rules can be taught. But although at this stage they do not exist in articulated form, they nevertheless do exist in the sense that they govern action (p. 73).

individuals often builds upon the deeds of greater predecessors in order to reach greater heights: "History looks backward into the past, but the lesson it teaches concerns things to come. It does not teach indolent quietism; it rouses man to emulate the deeds of earlier generations." One reason why newcomers are motivated would be partly due to respect and admiration of great developers, living, deceased, or fictional. The fame created by fake or real hackers in movies, stories, or news reports inspire aspiring hackers. Behavioral patterns and obligations spread with each interaction. The spontaneity of social order for OS and other social communities is voluntarily accepted.

Hackers are often motivated by intrinsic benefits. Intrinsic motivation includes the desire for autonomy and the feeling of competence rather than the external motivation of money and recognition (Deci, 1975). Hackers regard programming as an art form and thereby gain artistic satisfaction, similar to musical composers and artistic painters. Incentives matter of course, but the best work done are usually by those who are motivated by something other than external benefits. Moglen (1999) wrote, "It's an emergent property of connected human minds that they create things for one another's pleasure and to conquer their uneasy sense of being too alone." Free men invent; free men creates. Lakhani and Wolf (2005) found the sense of enjoyment gained from programming creativity was more powerful than extrinsic sources. The use of extrinsic sources, or motivations, is used most often On some level, however, all motivations, previously labeled 'intrinsic' or 'extrinsic', are fundamentally intrinsic, as the motivations always comes subjectively from within the individual. If one wishes to differentiate between what they deem as intrinsic and monetary motivations, the use of the term 'monetary motivations' is an apt description, without muddling and causing a

nonsensical distinction between intrinsic and extrinsic motivations.[48] As Olson (1965/1971, p. 61) wrote, "for social status and social acceptance are indivdiual, noncollective goods." Of course, motivations undoubtedly vary between individuals. Students and hobbyists were more likely to be motivated intrinsically while salaried and contracted programmers were motivated to sell related products and services (Hars & Ou, 2001). Furthermore, partaking in a "gift economy" (Mauss, 1950/2002) allows hackers to maintain social links and to encourage reciprocity, strengthening existing relationships but also opening up a path for strangers to get involved. Just as Coase (1937) argued firms reduced the organizational and negotiating costs of resources and contracts, the hacker community further reduces the cost of defining, monitoring, and enforcing licensing and other rights, as it creates a shared state of understanding to begin with. Intellectual property, then, has little or no hand in the space of software development, with some finding it as nonsensical as "proprietary mathematics" (Moglen, 1999). If a project starts to dwindle in developers and/or funding, awareness may grow that the project is creating and contributing something useful. Others may compensate or volunteer to work on the project (Hippel, 2017).

Hackers receive credit in several ways. Firstly, there is usually a commit log which lists all the contributors, no matter how few lines the submission. There is a norm against removing someone's name from a project without that person's consent (E. Raymond, 1999/2002). Secondly, some developers incorporate their own credit file within the package of the software. Thirdly, once a hacker has been around the OS scene for a while, other hackers,

[48] The problem of motivation explored in this thesis is not an issue with the Austrian branch of economics. Individuals volunteering for OS development rather than working for a proprietary firm is based on the individual's subjective valuation of differing ends. There is no distinction in between these two human actions, just as there is not such distinction between economic and non-economic action. As explained by Mises throughout his lifetime in many of his writings (and to give but one example: (Mises, 1981, p. 61)), "Economic action consists in the endeavor to remedy the state of dissatisfaction or, expressed differently, to satisfy wants as far as the scarcity of means allows. [...] Modern economics makes no distinction among ends because it considers them all equally legitimate, even those that the older view and the popular mode of expression regards as non-economic." Even if all goods were free (of zero price, no transaction costs, etc.), individuals would still have to economize on how they spent their time, since every action taken is another, different action not taken. Economics deals with exchange of scarce resources, of material and immaterial goods. Indeed, there exists individuals whose ends are beyond material things like higher incomes, though it is interesting to explore further the reasons why, as will be continued to be done in this thesis. It can be said, though, it is easier to understand monetary reasons for action while more intriguing is the ideological motivations. still, some may be tempted differentiate between intrinsic and monetary motivations, since money for many cases is used for other ends. That is, money is not an end in and of itself. Yet that would simply mean money is a tool for other intrinsic values, not that it is somehow different than intrinsic values. Intrinsic motivations, too, can lead to other intrinsic motivations. And there are still some who hoard money for the sake of hoarding money. In short, on the count of sounding too reductionist, all of human action is intrinsically motivated.

those who have been around longer, will take notice and understand his skill level and the type of work he or she does. The status and recognition is attained by community consensus. There is a mental record of a pecking order of hacking skill.[49] Those who send in smaller patches are usually not looking for this sort of recognition but simply want to give back to the community which has helped them in some way. Hackers, therefore, use their real names, tying their identities into the community—they do not use clichéd code names as commonly assigned in Hollywood movies.[50] Though only a few reach fame into the mainstream, such as Linus Torvalds, hackers are much more concerned about reputation within their own communities. Most problems or ill intent are solved through community vigilance and monitoring. Voluntary cooperation is seen as the most efficient way of organization.

To further protect OS and Linux, firms and volunteers have formed an alliance. One such alliance, the Open Invention Network (OIN), was founded in 2005 by IBM, Red Hat, and three other corporate Linux patrons. Members now include Google, Philips, and Toyota. There are also over 2,600 licensees who agreed to refrain from patent aggression (i.e. filing patent lawsuits) against OS developers, users, and distributors. They also exchange between each other free licenses for OS software. Proprietary firms that portray themselves as OS friendly or hire employees who contribute to OS projects (tracked on Github, OpenHub, and other project indexes) are applauded and celebrated in journals, videos, forums, social media, and blogs following the OS scene.

The maintenance of this culture, though costly, mitigates the other costs typically associated with firms, including moral hazard, principal-agent problem, opportunism, and shirking. When community trust and the culture begins to loosen, however, firms, as traditionally, will be the mode of organization which transaction costs will be the lowest in order to coordinate resources.[51] One Unix-like OS community Debian went so far as to create a social contract to

[49] Skill appraised as a combination of the beauty, simplicity, and elegance of the code (albeit a somewhat subjective determination) and the deftness in which hackers navigate computing resources (which includes disk space, memory, compilation time, computer power, etc.). Mathematical proofs are evaluated similarly. Elegance in simplicity.

[50] Today hackers are divided into "white hat" and "black hat" hackers. White hat hackers are ethical computer specialists who tests computer and encryption systems for any vulnerabilities in order to solve them before they are exploited. Black hat hackers are unethical hackers who exploit vulnerabilities for maliciousness or personal gain.

[51] Another high trust community involves the organization of the kibbutz in Israel. The scalability of the community, however, never reach over 10 percent of the population in Israel, and has been consistently dwindling over the past few decades. For further information, see Abramitzky (2018).

prioritize the rights of users, with those who adhere to the principles to not object or place roadblocks—legal or normative—to the distribution of free and open software.[52]. Debian has created a social contract in which the individuals are expected to enforce upon themselves, as they joined entered into the contract voluntarily (Rousseau, 2002). Explicitly written rules, either designed as a social contract or simply agreements, are similar to constitutions, in a manner that guides general behavior (see Harris (2018) for constitutions in peer-to-peer communitities and Leeson and Skarbek (2010) in criminal organizations).

Benkler and Nissenbaum (2006, p. 408-409) argued the virtues of OS communities include sociability, camaraderie, friendship, cooperation and civic virtue. Additionally, Benkler and Nissenbaum (2006) found the participants embody the "liberal virtues" of voluntary associations, promotion of common ends, and, if I may add, the focus on individuals and autonomy (both for the users and the developers). Therefore, though OS software is available to everyone as a public good without conditionality (by definition), OS communities create other goods and motivations which benefit only the individual contributor in order to promote reciprocal behavior. Indeed, as Olson (1965/1971) argued, "large organizations that are not able to make membership compulsory must also provide some noncollective goods in order to give potential members an incentive to join" (p. 16). Put differently, the benefits of OS software is inclusionary—that is, whoever wishes to join may do so[53]—while the benefits from joining a community and contributing is exclusionary—that is, only applicable to the individual contributor. Thus OS incorporates concentrated benefits as well as dispersed *benefits*.[54]

III. Firm and Open Source Software Development

In this section, the differences, similarities, and relationship between individual transactions (markets), firms, and OS will be explored. For the majority of the section, I will highlight

[52] The Debian community ratified version 1.0 on July 1997 and version 1.1, which supercede the previous version, on April 2004. For the entire contract, see https://www.debian.org/social_contract (Last accessed November 14, 2018).

[53] Additionally, building on Olson's (1965/1971) definition, an inclusive collective good is "such that the benefit a noncooperator receives is not matched by the corresponding losses to those who do cooperate" (p. 40).

[54] Success, however each culture and individual defines this, will breed envy since this is a necessary human trait (Schoeck, 1966/1987). An interesting research topic would be examine and analyze the structures involved in OS communities to mitigate envious behavior, between skilled developers and between different less and more successful communities and projects.

the advantages of proprietary software development, namely greater control, stricter deadlines, and more software support (subsections A and B). Furthermore, I will analyze the effect firms have on transaction costs and how OS attempts to deal with transaction costs. It will indeed be interesting to see how each organizational structure and a mixture of both adapts to the ever-changing needs of users. In the third subsection, the distinction between users and developers, it will soon be obvious, is not as clearly divided as in firm or market structures. The main reason being OS entices users who are already more technologically savvy. The last section will attempt to put together the previous three subsections and answer the follow two questions: (1) why firms fund and create OS projects and (2) how the relationship between firms and OS is maintained.

A. Advantages of Software Development in Firms

Ronald Coase in two seminal papers asked firstly, why individuals traded within firms, rather than on the open market guided by the price mechanism (1937); and secondly, why there were private provision of public goods—in his case, lighthouses (1974).[55] Indeed, Coase questioned the existence of firms if markets were so efficient at conveying relevant information through prices. The reason firms form and cluster resources, Coase went on to answer his own inquiry, is ultimately due to transaction costs.[56] When the transaction costs of defining and enforcing property rights, allocating resources, and coordinating efforts are low, individuals are able to exchange. On the other hand if the opposite is true, firms form. In his second paper, responding to Mill,[57] Sidgwick,[58] and Pigou,[59] Coase found that private interests continued to build and maintain lighthouses even when all the benefits could not

[55] Additionally, Block and Barnett (2009), Laurent (2003), and Zandt (1993) supported Coase's conclusion that private provision of lighthouses exists. In some cases, private provision was crowded out by subpar public efforts Candela and Geloso (2018). Lastly, Taylor (2001) argued it was morally wrong that private profit was made at public expense.

[56] Contrary to many scholars in the field of law and economics, Coase did not advocate imagining a world of zero transaction costs and then resolving judicial and property rights cases as such—the so-called "Coase theorem." Nor did he mean to say it does not matter how to distribute property rights since the most efficient outcome will occur anyways. Such worlds do not exist and are impossible, see "Eric Raymond on Hacking, Open Source, and the Cathedral and the Bazaar" (2009). Instead, while responding to the Pigouvian argument to simply tax away negative externalities or subsidize positive externalities for optimal outcomes, Coase emphasized the role of tradeoffs per transaction costs. That is, how could property rights disputes and externalities otherwise be solved without a Pigouvian tax or subsidization. It is precisely because there *are* transaction costs that the distribution of property rights be carefully considered.

[57] Mill, 1848/1909, book V, ch. XI, sec. XV.

[58] Sidgwick, 1887, book III, ch. II.

[59] Pigou, 1920, ch. XIX.

be internalized. Put simply, Mill, Sidgwick, and Pigou argued no sane entrepreneur would undertake the business of this ruinous business model. Governments must subsidize or operate to supply this useful but unprofitable service. In respect to OS, the same question is asked: why would developer entrepreneurs endeavor this seemingly unprofitable business? If markets and firms are conducive to efficient coordination and transactions, why does OS development exist—and does it need to exist? The answer, simply put, again, is transactions cost. However, further explanations are necessary, rather than this general—and generic—answer.[60] Just as a "worker is not motivated to move from department γ to department χ because of a change in relative prices" (Coase, 1937, p. 387), OS developers do not decide to contribute because of a change in relative prices for different software and different projects. OS developers do not perform transaction costs in their heads to decide whether or not to contribute, not in the least the founder(s).

Proprietary software retain advantages over OS software in several ways: 1) specialized development, 2) software support, 3) usability, and 4) security. In OS, the project may evolve differently than what may suit the needs of other customer firms. Proprietary software, in this sense, is able to develop software towards more specific uses and needs, in respect to the requests of their clients and other customers. In proprietary software, scheduled deadlines allow for the timely release of the software. While in OS software, there are also release dates, more often than not, these release dates are general guidelines. Software submissions in OS happen when volunteers decide on their own while submissions in firms are designated and scheduled. OS projects can decide to release their software on different channels, which may include alpha, developer, beta,and stable—or just stable. The alpha channel is usually the most unstable, with the bare minimum to get the program running, very few features, and a conceptual rather than a concrete program. The developer (sometimes shortened to dev) channel is the second most unstable, with a high degree of crashes and frequent bugs, but usually the most frequently updated. Here the latest features are implemented in order to receive the quickest feedback. The beta channel generally has the major bugs and crashing instances fixed, therefore lowering the risk of losing work and frustrating bugs, though the program is not quite ready as a daily driver. The stable channel, of course, is the one ready

[60] Just as many economic answers on pricing can be answered as "supply and demand," such a brief phrase remains unsatisfactory in explaining, in detail, the true causes.

for release and public use. Many projects, like the Chromium browser[61] and Chrome OS, allows users to easily switch between beta, developer, and stable channels. As such, version releases of OS software are typically early and often.

Issue-trackers and version control systems annotates the early-stage project in order to receive feedback on features and bugs. The final release date is unknown whereas proprietary software is more constricted. With leadership and explicitly defined goals, proprietary software development is more coordinated in the sense that every programmer has a role but also more rigid. Having a full dedicated team behind a project can be superior in terms of longevity of the project and continued service. In OS software, updates and features are implemented usually at a faster rate than proprietary software, though this is dependent on the number of volunteers in OS and the complexity of the project. With larger funds to draw from proprietary software is able to research and build more complex projects, including developing software and hardware alongside one another, creating better compatibility. In the case of Google and Microsoft, both companies have recently released phones and laptops with hardware they built themselves paired with its software developed in house, in order to deliver a faster and more fluid user experience. The usability of proprietary software in terms of the graphical user interfaces (GUI), animations, and overall aesthetic is, for the most part, although subjective, better when compared to OS software. Proprietary software such as Spotify, the Microsoft Office suite, MacOS have designs that are more cohesive and user-friendly. Lastly, since the source code is by definition closed, exploits and weaknesses are difficult to locate and the source code is difficult, or impossible, to duplicate. Of course, no software—open or proprietary—will ever be perfectly secure, as what is built by humans can be dismantled by humans. But unknown source code is more difficult to invade.

Software development suffers from two main issues: required expert knowledge and knowledge of the preferences and uses of the software. Users and developers use programs differently, since they have differing expectations on the limits of a program. Different bugs and malfunctions may arise out of different users due to the varied understanding of what the software can do. These users may use inputs the creators of the program never intended. The program may be used in conjunction with another program, causing certain incompatibility issues. Morever, as Henkel (2003) noted, manufacturers and OS developers tend to focus

[61] The OS version of Google's popular Chrome web browser.

on different innovations; manufacturers (firms) tend to develop software many will want while OS develoeprs tend to create software for what they or a few will want. Some users would like more features than provided for by proprietary software. Linux, for example, allows broad customization in appearance, network security, default applications, and many others. Proprietary software is able to accumulate plentiful expert knowledge through hiring employees and such software does indeed satisfy many, if not a large majority, of the users. However, there remain the preferences of users regarding privacy and compatibility. Some users are incredibly sensitive to how companies and software may handle their personal information, therefore they prefer software in which the code is open and accessible, allowing them to inspect and understand the code for themselves.

B. Disadvantages of Software Development in Firms

By attempting to centralizing information, proprietary software runs into a problem: there is a loss of localized and tacit knowledge (Hayek, 1945). Individuals, not point out a banality, "learn by doing" (Arrow, 1962), and certain tasks cannot be taught by speech or writings to another individual. Some knowledge is gained only through physical action. The specificity of every human and material resource and opportunity costs is lost or at the very least, diminished. Another problem is the decreased adaptability of firms, though large OS communities and large firms suffer from this problem as well. Engaging with large amounts of employees and resources increases the time it takes to transport information and to redirect goods and labor. Moreover, the incentive schemes in firms are complex, with each individual having different goals and interests. Where one employee may become motivated through monetary compensation, another may otherwise seek personal fulfillment. In other words, though firms decrease certain transaction costs, the "internal organization brings another kind of transaction cost, namely problems of information flows, incentives, monitoring, and performance evaluation" (Klein, 1996, p. 6). For OS software development, volunteers may also have unknown and complex reasons for participating, but motivation is a much less problem as by submitting patches they have, by their actions, proven their want to contribute. There is a strong incentive for voluntary contributions in order to satisfy specific needs. Moreover, as mentioned earlier, talented programmers may wish to signal

their skills, or perhaps simply find pleasure in working on an interesting OS project rather than perform routine work from an employer. A study conducted by Grewal, Lilien, and Mallapragada (2006) found technically challenging OS projects attracted more and better skilled programmers.

Transaction costs are not homogenous. While transaction costs for collective action have decreased in terms of cheaper computing and communication costs, transaction costs may increase in terms of intellectual property enforcement. That is, in order to use proprietary software, compliance in use and fees are required. Property and contracts make resource and labor allocation less adaptive. An experimental game conducted by Crosetto (2010) suggested stronger copyright and patents, among other intellectual property protection, will result in less innovation and therefore incentivizing developers to find alternatives to code development, leading to a growth of copyleft and the OS commons. The employees of the firms will more likely work and collaborate within the firm even when it would be more advantageous to collaborate with outsiders. Firms must expend effort and money in order to absorb useful ideas and conventions from sources external to the firm.

In terms of hardware compatibility, some users have specific equipment or old hardware where proprietary software either runs slowly or is not even supported. The need to compile packages for a particular type of hardware is in demand. For example, in the case of Linux, it is run on computers in NASA and the Large Hadron Collider (LHC) and can be made to run on the iPod classic and even the Gameboy Advance, hardware which was never originally intended but has been adapted for. The flexibility of open code and community involvement in plenty of cases surpass proprietary code. The reason being while companies do pay for specific software, OS allows individuals to incorporate hardware compatibility on their own. Moreover, OS is less likely to deny incorporating a compatibility issue where the only a few users will benefit, since these users can easily create and submit their own patches.

C. Beyond Firms: Open Source

Firm size growth will level off and stagnate when the internal complexity causes higher costs than what a smaller firm would incur for the same marginal result. Rather than the traditional two decisions of making what the company needs internally or buying what

they need externally, another option has arisen: outsourcing to a third party. Outsourcing is different from purchasing goods and services externally in that the third party technically works for the company through a contract or agreement (typically for a limited duration). OS is a subset of outsourcing. Firms can start or finance an OS project. This alone, however, does not imply an outgrowth of OS. Without being able to internalize benefits there is difficulty in providing a good or service. By providing a good which anyone with access benefits from, it seems difficult how to solve the problems of free-riding and the tragedy of the commons. Indeed, the argument of common ownership resulting in poor care traces back to Aristotle.[62] It is impossible to incentivize widescale economic activity through good will alone. Benkler (2002), therefore, used both Coase's theory of the firm and Demsetz's theory of property rights to explain why OS production emerged (Table 3.1). Benkler categorized three idealized concepts of organizing exchange: (1) individual markets, (2) organizing markets, and (3) peer markets (or exchange). Throughout this thesis, we asked why do individuals engage in either individual transactions, firm production, or OS production.

Benkler explained the likelihood of six production modes determined by the existence of a property rights system and the implementation cost of that property rights system (Table 3.1). If individual markets are more efficient and a property system is more enforced, then markets will form. However, if the implementation costs are higher, then commons will form.[63] If organizing exchange is more efficient than individual or peer markets and a property system is more enforced, then firms will form. Though if implementation costs are high, then more localized common property regimes (in Ostromian terms, common-pool regimes) will form. Lastly, if peer exchange is more efficient than market and organizing exchange and a property rights system is more enforced, then a system between that of firms and OS will emerge. On the other hand, if implementation costs are high, then pure OS (peer production) will emerge. Implementation costs here are, includes but not limited to, codifying and enforcing property ownership rights.

[62] "What is common to the greatest number gets the least amount of care. Men pay most attention to what is their own; they care less for what is common; or at any rate they care for it only to the extent to which each is individually concerned. Even when there is no other cause for inattention, men are more prone to neglect their duty when they think that another is attending to it" (Aristotle, 1998, book II, pt. III).

[63] Benkler's use of commons here is different than of CPR. Property rights, if I may add a reminder, is a spectrum of ownership regarding disposal, exchange, modification, and other maniuplations of property. Here, commons refers to resources that are not restricted and managed by individuals or a community whereas CPRs manages resources within a community.

Table 3.1 Ideal Organizational Forms as a Function of Relative Social Cost

	Property Rights System Enforced	High Implementation Costs
Market exchange > Organizing or peer exchange	Markets	Commons
Organizing > Market or peer exchange	Firms	Common property regimes
Peer exchange > Organizing or market exchange	Proprietary open source	Peer production open source

(Source: Benkler, 2002)

In the book *Free Innovation* Hippel (2017) challenged the standard views of the producer innovation paradigm (Baumol, 2002/2004; Romer, 1990; Schumpeter & Swedberg, 1942/2003). Producer innovation has compensated transactions at its very core. The concept of free innovation is a novel product, service, or process that is developed by consumers through their own private means (not being paid) and not protected, therefore the innovation can potentially be acquired for zero price (Hippel, 2017, p. 1). Though contrary to Demsetz's claim of property rights arising out of ease of enforcement and internalization of benefits, though software patent has become the norm since 1996 (Tsai, 2008), OS licensing treads against this current. Hippel argued free innovation is fundamentally different than the producer innovation paradigm, that the two cannot be incorporated, though both compete and cooperate in complementary ways.

The reason why OS succeeds in this manner is by a particular set of circumstances: (1) the capital goods required to do the work are cheap, (2) the limiting factor to the work is human creativity and attention, (3) the work is intrinsically rewarding, and (4) there is an objective metric for success ("Eric Raymond on Hacking, Open Source, and the Cathedral and the Bazaar", 2009). Without an objective metric for success, there is no agreed upon value of submission; the infighting and debating involving whether or not the contribution is valuable and useful will lead to stagnation, limiting the scalability of the project. The objective measure of success in the case of OS is quite simply whether or not the code compiles and does what the creator(s) intended.

With the physical capital goods becoming cheaper (e.g. computers, network towers, servers, etc.) and the declining cost of communication, OS will continue to be competitive

against firms the more valuable human creativity becomes. Benkler (2002) called OS development "peer production." Peer production, he found, is better suited at identifying and allocating human creativity to work on information and cultural resources (Benkler, 2002, p. 375). At first early in human development, there were individual transactions on the market. As the benefits of firms and grouping resources together were realized and turned to be more profitable, entrepreneurs attempted to organize resources and labor in a more productive manner (with the outputs yielding greater benefits than the sum of the inputs). Now, it seems that since firms have reduced transaction and communication costs, that entrepreneurs are now seeing opportunities to become involved in OS development, peer production (Benkler, 2002), the gifting economy (Mauss, 1950/2002), the sharing economy (Lessig, 2008; Munger, 2018),[64] and other organizational structures in a knowledge society. The terms "gifting" and "sharing," in this instance, will be used interchangeably and defined as individuals or groups who give but do not directly, immediately, or obviously benefit as a result (McGee & Skågeby, 2004). The free-rider problem in a gifting economy, unlike in a market or a firm, is not a fault but an essential feature. In a sense, unlike academia and the arts, plagiarism is encouraged and those who wish to copy and plagarize must allow their works to be copied and plagarized as well. Individuals may have created the product to help themselves and will freely release this information because one does not lose anything, so that others may benefit without outright demand for payment or reciprocation. Though gifting remains difficult to explain fully in economic or sociological theories, which, as McGee and Skågeby (2004) summarized, include pseudo-gifting, socially enforced norms of gifting, ideological gifting, and altruistic gifting.

Though software, when simple and short, can be verified through proofs and a small group of developers, on a larger scale, for example Linux nearing 20 million lines of code, more eyes on needed to spot errors. With OS, bugs are less of a problem, even without a direct goal of limiting bugs, because more users and volunteers will report and fix bugs. E. Raymond (1999/2002) was responding to Brooks Jr. (1975/1995) who argued that due to the complexity of software development, that it must be a tightly planned and controlled procedure. The goal and programming process must be clearly defined and long development times used to ensure the least amount of possible bugs.

[64] Also otherwise called the gig economy or the hybrid economy.

The theory of sticky wages posits worker's earning do not adjust quickly to market demand due to a number of factors. This relates back to Stigler's (1961) notion of search prices. The cost of finding relevant information results in "stickiness," or in other words, the loss of innovation is a function of information transfer costs. Hippel's (1994) summary of Rosenberg (1976) and Nelson (1989) highlights Hayekian arguments (Hayek, 1945): calling much of technological knowledge costly, difficult, consisting of "innumerable small increments and tacit" (Rosenberg, 1976, p. 78). Regarding information, Hippel (1994). On a more local level, with closed source software, consumers get used to an ecosphere. For instance, because Microsoft, Google, and Apple Inc. share limitedly their source code, consumers must choose which system to learn and spend most of their time and money on, as these companies provide platforms for which their own apps are easily supported but not others (e.g. Google Playstore and Apple iTunes store). Each companies' interface and hardware are different and relearning different sets of features and software takes time. OS caters to end-users who are primarily tech-savvy. Interestingly, Linux users sometimes engage in the behavior known as "distro-hopping," where they frequently change the type of Linux distribution (operating system) on their machine. It is due to the fact that most Linux users are tech-savvy, though, at the same time, free and open access to software that creates a feedback loop to this behavior.

D. Symbiosis of Firms and Open Source

The physical capital of the telephone, radio, television, and other infrastructure, along with proprietary communication software, has decreased the cost of computation, communication, and organization enough for OS software to arise. Though it seems OS and proprietary development are polar opposites, there are areas where the two have cooperated and even merged, where I make less of a distinction between the two as Hippel (2017). As noted earlier, many OS software was first initiated and developed by companies but transitioned into OS software. The Internet, many computer programming languages, and other OS software projects are developed in academic research facilities intended for corporate use. While some users are fanatical about OS software to the point that is hostile to proprietary software and refuse to use any, fearing privacy concerns and the closed nature of the software, several firms have sought to commercialize OS software projects, such as operating systems Red

Hat[65] and Novell. We see how, and I repeat, firms and OS are competitive in certain areas and cooperate with others. While some OS communities have incorporated in order to protect themselves from individual liability, the division between proprietary software and OS software is not so clear-cut. Schumpeter (1983) distinguished between the birth of a new idea, method, or product (invention), the economic application into a new product or process which lowers transaction costs (innovation), and the gradual adoption by consumers and businesses (diffusion). Firms have lowered hardware, infrastructure, and communication costs. Every transaction has transaction costs for both the buyer and seller, in markets, individual exchange, or OS. Some of the these transaction costs include *ex-ante* search, information, and bargaining, while *ex-post*, includes control and contractual enforcement costs. The main obstacle for the diffusion of proprietary software remains in price and intellectual property protection. OS, on the other hand, is able to freely diffuse, though search costs remain.

In fact, some have indeed argued most of the innovation and developing capacities enabling OS software originated as side projects from firms (Mehra, Dewan, & Freimer, 2011). The Internet, Linux itself, many computing languages, and other projects were developed in either an academic or a firm setting. Akin to the Soviet Union, when abolishing prices within the country and relying on pseudo-prices from the international market (Mises, 1949/1998, ch. XXVI),[66] a plausible argument could be made that OS software relies on the backbone of the prices and discoveries made by proprietary software. Developers have access to the market prices of proprietary software and therefore can price their software or suggest donations

[65] Red Hat founded in 1993, is one of the first professional OS business model. The company includes professional quality assurance and subscription-based programs. These programs support, train, and integrate services that aid in customer usage of the OS software products. Additionally, they contribute code and donate to other OS projects, including the Linux kernel, LibreOffice, and the GNOME desktop environment. In 2012 Red Hat became the first OS company to surpass $1 billion revenue in a fiscal year, surpassing $2 billion in 2015, and nearly $3 billion in 2018. They also supply cloud services and other technologies used in data centers.

[66] Furthermore, Rothbard wrote, "The extent of socialism in the present-day world is at the same time underestimated in countries such as the United States and overestimated in Soviet Russia. It is underestimated because the expansion of government lending to private enterprise in the United States has been generally neglected, and we have seen that the lender, regardless of his legal status, is also an entrepreneur and part owner. The extent of socialism is overestimated because most writers ignore the fact that Russia, socialist as she is, cannot have full socialism as long as she can still refer to the relatively free markets existing in other parts of the world. In short, a single socialist country or bloc of countries, while inevitably experiencing enormous difficulties and wastes in planning, can still buy and sell and refer to the world market and can therefore at least vaguely approximate some sort of rational pricing of producers' goods by extrapolating from the market. The well-known wastes and errors of this partial socialist planning are negligible compared to what would be experienced under the total calculational chaos of a world socialist state" (Rothbard, 1962/2004, p. 959-960).

within that reasonable range.[67] OS software also depends and copies on the initial types of software first created in a proprietary manner. Operating systems, word processors, video editing, and many of the programs were already created by proprietary developers through firms and the academic setting; OS simply recreated the previous software in an open manner and, for many of the software, making it publicly available at zero price. At least in this aspect, OS can be viewed as imitative, though—and not in contradiction—also entrepreneurial. Put shortly, software and related services were already created by firms in a proprietary manner, and the prices of these goods and services were already discovered in the market process.

Firms encourage programmers to work on OS projects in order to anticipate the strengths and weaknesses of different approaches (J. Lerner & Tirole, 2005). Moreover, companies have continued to offer incentives for programmers to work on both proprietary and OS projects, and if both projects provide profit, then companies offer bonuses for both projects simultaneously. Corporate contributions in OS projects have increased both in funding and the creation of new projects. One of the reasons, briefly mentioned by Kumar, Gordon, and Srinivasan (2011), is multiproduct firms may wish to fund OS projects in order to weaken a competitor in another market.[68] Though some companies remain opposed to OS, like Microsoft historically,[69, 70] other companies, like IBM, have helped develop services and products complementary to OS product, resulting in OS and companies "living symbiotically" (J. Lerner, Pathak, & Tirole, 2006).

[67] The formulation of prices is not a result of a static equilibrium, but as a result of the similar opportunities, events, and incentives of the previous day. As a result, economic problems arise only from a consequence of change. For further readings on equilibrium, price theory, and the calculation debate; see Hayek (1940), Mises (1951/1962), Salerno (1995).

[68] Kumar et al. (2011) went on to suggest this would explain why IBM contributes to Linux or why Sun Microsystems (acquired by Oracle Corporation in 2010) contributes to OpenOffice: in order to reduce Microsoft's market share in operating system and word processing.

[69] The antagonism between Microsoft and Linux spans over two decades. In October 1998, a leaked internal memorandum from Microsoft highlighted Microsoft's worry and strategy against OS development, noting "OSS poses a direct, short-term revenue and platform threat to Microsoft, particularly in server space. Additionally, the intrinsic parallelism and free idea exchange in OSS has benefits that are not replicable with our current licensing model and therefore present a long-term developer mindshare threat" (Source: http://www.catb.org/esr/halloween/halloween1.html, last accessed November 10, 2018). Past Microsoft CEO Steve Ballmer called Linux a cancer (Greene, 2001) and likened it to communism (Lea, 2000).

[70] Though in October 2018, Microsoft announced it will be joining the Open Invention Network (OIN), the community created to defend Linux developers, distributors, and users from patent trolls and companies antagonistic to Linux and OS. In doing so, Microsoft made over 60,000 of its patents OS in order to further protect the Linux development community from further patent infringement lawsuits. The reason for this is that a large part of Microsoft's revenue is now derived from cloud computing, which is intrinsically linked with Linux and cannot be provided completely in a proprietary manner. Indeed, cloud computing and cloud database cannot be disentangled from OS software. Microsoft has even scrapped the propriteary nature of their web browser, Microsoft Edge, now building off the OS backbone of Chromium.

OS software is not lucrative by itself and as a result, is often bundled with other goods and services (Maurer & Scotchmer, 2006). Not all users of goods and services need to be charged in order for the goods and services to be profitable. Bundling is used to increase revenue, reduce costs, or to deter new entry (Zhu & MacQuarrie, 2003). For OS, the first two reasons are true but instead of deterring entry, most OS software is bundled into other proprietary software to encourage entry and user adoption. Changes in digital bundling, Zhu and MacQuarrie (2003) further explained, is a response to secure profits from a changing technological landscape. As Potts (2012) wrote, bundling unknown products with known products reduces the transaction and search costs while increasing the discovery of good novelties for adoption. The overall bundle of products and services may be valued more highly than each separate product/service. Engelhardt (2011) listed various industries with joint ventures in the OS software business model, including computer system vendors, telecommunications vendors, microprocessor producers, and other development and supporting companies. Complementary products and services are able to be bundled in order to enhance customer satsifaction. Examples include bundles of OS software with hardware (e.g. laptops and computers preinstalled with Linux and phones with Android), security services (e.g. Red Hat), together with other more popular OS software (most applications in Linux operating systems are OS themselves), and with proprietary software. One of the reasons why OS can operate with limited or even no price mechanism, I may surmise, is because of the decreased transaction costs, the mistakes are less costly and therefore experimentation will not result in as high cost of mistakes as in traditional firm. The problem of the optimal product diversity for social welfare has concerned economists since Chamberlin (1950). By bundling OS software with proprietary software or other products, this in effect creates new innovative products, increasing the heterogeneity of goods available, while at the same time creating a working price mechanism in order to better evaluate profit and loss with experimentation. In this manner, most OS is distinct from a non-profit organization, who are a "sovereign unto themselves" (Mises, 2008) and are limited only by the amount of capital at their disposal, in an attempt to defy the wishes of the public. OS communities and organizations still work to satisfy public needs albeit to a different degree compared to traditional markets and firms. A. Martin (2010) explored various feedback mechanisms (in his case, the satisfaction of public

needs) going from tightest to loosest: sole proprietorship, corporation, private charity, local government, and the bureau. This means depending on the OS community the appropriate feedback ranges similarly between a sole proprietorship to a private charity.

Lessig (1999c) argued just as there is a mix of community and privately owned spaces in the physical space, there will be a combination of open and proprietary software in cyberspace. In a related study, Gaudeul (2008) compared three archetypal models: pure OS software industry, pure closed source software industry, and a mixed industry. He found the welfare in the mixed industry higher when compared to the pure models. IBM has heavily invested in Linux, Apache, Eclipse, as well as many other OS projects.[71] Apache-related products provides hosting services to around 40-50 percent of all websites on the Internet.[72] Since IBM's main source of revenue is derived from selling web servers, which will benefit from improvements to Apache. Even code developed in-house is released to Apache. Furthering the point on the dependency between OS and traditional firms, on October 2018, IBM announced it will be acquiring Red Hat for $34 billion.[73] Both companies have ensured the public and the OS community the acquisition will continue the growth and innovation in cloud computing and other OS technologies. Similarly, as Microsoft's business model incorporates more cloud computing and becomes more open, cooperating with Linux, Microsoft has released all of its patents—over 60,0000—to OS. Microsoft now has over 2,000 OS projects, some small and some large.[74] In 2017 on Github, Microsoft and Facebook projects had the highest contribution rate.[75] The reciprocal nature of OS volunteers, it seems, is not limited to open communities but extends to proprietary firms.

The physical capital costs becoming fixed and increasingly cheaper, information has become freer and the most valuable resource, human creativity, has been able to come together more fluidly. Firms are starting to realize greater profits can be made by supporting

[71] Official statement: https://www.ibm.com/analytics/us/en/services/open-source.html (last accessed: October 29, 2018).

[72] One daily updated report indicates Apache is used by 45.3% of all websites as of October 2018 (see https://w3techs.com/technologies/details/ws-apache/all/all). The methodology included was the investigation of technologies used by websites, not webpages; the sample involved the top 10 million websites in order to limit the inclusion of domain spammers; and did not define subdomains or redirected domains to be separate websites.

[73] This is by far the biggest IBM deal in its history—and the third biggest in U.S. tech history. The other two largest deals were the $67 billion merger between Dell and EMC in 2016 and the $41 billion JDS Uniphase acquisition of SDL.

[74] See further: https://opensource.microsoft.com/ and https://github.com/Microsoft

[75] https://octoverse.github.com/

rather than being against OS. Less software that is developed is actually sold, the vast majority of software development is to solve internal problems within firms—to solve business problems. Perfect information is impossible to acquire; the mixture between closed and open development must balance between the lost of exactness of the information over the cost of obtaining more. At the same time, firms continue to compete with each other and other OS ventures; in the act of competition, coordination appears. Though firms produce varying goods in different features, certain standardization of products and their features arise. Through standardization, communication and collaboration become easier, therefore creating a suitable environment for large-scale projects. Haruvy, Sethi, and Zhou (2008) suggested firms adopt some form of OS development if bug reports are critical to the project but the effectiveness of in-house programmers is low, in-house production is costly, or initial quality is low.

The examples of reputation and branding, community building (or in other words, social capital), and other social learning mechanisms have, as Chamlee-Wright and Myers (2008) argued, generated surprising success in the feedback they provide. Though—and I would agree—Chamlee-Wright concluded, these non-priced mechanism cannot replace prices and calculability, even if they may mimic other attributes of market prices to allow for coordination and cooperation. Indeed, in her latest book *The Cultural and Political Economy of Recovery* (2010), she noted how prices (in her case study of New Orleans' efforts after Hurricane Katrina) can be distorted by public policy through introducing regime uncertainty and by disrupting signals emerging within commercial and civil society. In a related manner, the intellectual property system may distort the prices on the availability of knowledge and code, though this issue will be explored later in some detail.

IV. Dispersed Planning in Open Source

F.A. Hayek is useful for explaining why OS software have emerged and continues thrive. Relatedly, this builds upon the Coasean foundation, which should be of no surprise for anyone aware of the historical history between Coase and Hayek.[76] As Foss and Klein (2009)

[76] They were both colleagues during the 1930's working at the London School of Economics (LSE) during the 1930's.

and Langlois (2013) argued, the Coasean firm is consistent with the Hayekian knowledge problem, and with Hayek's thoughts on markets. Indeed, both analyze and address the limits of the firm and the market. Hayek's theory of dispersed knowledge in society tackles not only the economic problem of calculation and efficient allocation of resource but also the problem of knowledge and epistemology as a whole (Hayek, 1945). Furthermore, as Hayek points out, "Economics has long stressed the 'division of *labour*' which such a situation involves. But it has paid much less stress on the fragmentation of *knowledge*, on the fact that each member of society can have only a small fraction of the knowledge possessed by all..." (Hayek, 1973/2013, p. 14). Sites like Wikipedia or Quora, obtained great success from individuals with expert and specific knowledge to fill in missing gaps of information. Mises found planning omnipresent, "[E]very human action means planning. What those calling themselves planners advocate is not the substitute of planned action for letting things go. It is the substitution of the planner's own plan for the plans of his fellow man" (Mises, 1951/1962, p. 538). The confusion, Mises and Hayek added, is not choosing between planning or no planning; the question is who fundamentally makes the decision. How much information does one need in order to make an immediate decision and at what additional cost to obtain the next marginal bit of information? This economic calculation can, as Hayek and Mises argued, only be made through the price system.

The argument I made in the previous section is that OS still relies on the price system by incorporating some features of the firm and consequently bundling of products. In this section, I continue exploring this argument through a more Hayekian framework. The price Hayek argued "there is no need for the great majority of them even to know where the more urgent need has arisen, or in favor of what other needs they ought to husband the supply," (Hayek, 1945, p. 526) Though OS communities are, by and large, spontaneous organizations (more on this later), the purposeful action, the directing intelligence of individuals, particularly of the founder(s), creates the rules and turns attention towards which area to focus on for software development. Human action, while purposeful in that individuals attempt to move from a state of unsatisfaction to a state of greater satisfaction, necessarily encompasses abstract thoughts, since the mind cannot fully comprehend all of reality (Hayek, 1973/2013, p. 29). OS developers, in accordance with his or her capacity to move successfully in the world,

maintains such abstractions and heuristics, but must obey the feedback from his or her environment. Indeed, "we never act, and could never act, in full consideration of all the facts of a particular situation, but always by singling out as relevant only some aspects of it" (Hayek, 1973/2013, p. 30). Founders and contributors most likely, as explained earlier, have a propensity, or intuitive sense, for opening up coding and software information. OS communities attempt to impart, through blogs, advertisements, and word of mouth, the value for creating a more open and free foundation for sharing code. This is because the cost of spreading such a message, OS members find, is less than the benefit of creating a shared belief system. In other words, the price mechanism is simply a mechanism for communicating preferences within a society, and OS members wish to change the preferences of users.

On one end, decisions can be made by a centralized group who attempt to aggregate information, or on the other side of the spectrum, decisions can be made by local actors holding specialized and tacit knowledge. In OS, no single or group of programmers hold all the relevant knowledge about consumer preferences, the optimal utilization of machinery and human skill, and the correct management time—it is also not clear that if this was possible that it would be preferable. Indeed, when attempting to solve complex problems as a whole over a large region and/or long duration the epistemological problems can only be solved by approximation. The solution "is the 'unplanned' activity of a team of independent calculators each of whom limits his interest to the single centre of which he is in charge" Polanyi (1951/1969, p. 181). Uncertainty is ubiquitous for entrepreneurs and every human action. Sheer ignorance, the state of not knowing even the problem, is a state in which entrepreneurs attempt to navigate around (Kirzner, 1997). When engaging with reality from knowledge—theoretical or from experience—feedback mechanisms which let the entrepreneurial developers know which decisions are indeed suitable is of the foremost importance for coordination. Firms of course also attempt to solve this problem, with firms being able to grow or shrink in size or market-share by being able to provide consumers better goods and services in relation to their competitors. Change is a constant in reality and with it, the economic problem of scarcity (resources and time) will arise. Pertaining to economic planning and scarcity, OS, explored in this section, discover knowledge and coordinate action better than markets and firms in certain aspects, though as in all things

economic there are trade-offs.

In this section, as an overview, I begin with how the distinction between OS and firms, noted in the previous two sections, in reality is not so clear cut as I made it seem (in subsection A). There are aspects OS communities which incorporate firm-like attributes and vice versa. In doing so, programmers build off of the strengths and weaknesses of each organizational structure specific to their projects and members. In the second subsection (subsection B), taking a look at branding and reputation of OS projects and communities, I will explore how OS developers take advantage of their reputation to promote their projects. In the third subsection (subsection C), I will explore with how OS deals with the problem of dispersed knowledge, emphasizing knowledge overlap and redundancies, and the importance of experimentation. Closing this section, I will deal with the variations of hierarchical structures in OS (subsection D). For the most part, firms have a strict hierarchical structure whereby usually individuals attain adequate roles in relevance with their skills, this subsection will ask if OS, in the first place, contains hierarchies and if so, how they organize through a voluntary and evolutionary process.

A. Open Source Branding and Reputation

Stigler (1961) found the search for prices in a market exhibited costs—sometimes great costs. Furthermore, Akerlof (1970) suggested asymmetric information in terms of quality could lead to the breakdown of the market. Firms attempt to circumvent the asymmetric information gap by including branding (Roa & Ruekert, 1994), advertising (Butters, 1976; Conrad, 1982), warranties (Schwartz & Wilde, 1983), guarantees, and among other solutions which builds trust. Though firms have various methods and procedures to solve the issue of imperfect and asymmetrical information, it is not obvious how OS groups can do the same due to limited feedback from the profit and loss mechanism. The reputation of OS projects is not an aggregate of every partipants' reputations. Consumers rarely if ever know which developers contributed to the project and for the most part view the reputation of each project as an indivisible whole, not a lump sum.

Branding is a vital part of this solution. In addition to reducing search costs, brands help capture in short hand what the firm, organization, or group stands for, their purpose, methods,

and values. Pitt (2006) argued the evolutionary nature of brands has four phases. In the initial phase, brands benefit sellers very little and signal less information to the buyers. Oversupply is rare and buyers are not faced with enough choice to make distinctions between different products. In the second phase, entrepreneurs learn how to mark distinctions between each of their products, allow consumers to become more sophisticated and to make better decisions. The third phase of the evolution marks the strong brand. Search costs are mitigated as consumers now rely on their chosen brand. Sellers have earned a reputation for certain features, standards, and qualities of their product, resulting in more passive buyers. At the final phase, Pitt (2006) wrote, brands become nonproprietary, with sellers benefiting very little but a lot for the buyers. With many OS products, the differences between buyers and sellers, users and developers are blurred. In general, there are three types of agents for the purchase of goods and services: (1) consumers looking to purchase said goods and services, (2) producers who create it, and (3) an intermediary who makes the connection between consumer and producer (triangulation).

In actual markets, moreover, these categorization are often blurred as well, though in OS, this is blurred even further. The relationship between consumers and producers in many public economies become "co-producers." Parks et al. (1999) noted students may supply their own education absent teacher inputs, but teachers can supply little education without student inputs. Community policing would have little effectiveness if citizens did not have some level of self-protection without the police. Indeed, Parks et al. argued some sort of mix between regular and consumer producer input is necessary for rising service demands. In proprietary markets, a similar phenomenon is occurring, with Microsoft and Apple Inc. often referring to their customers as "creators" and marketing the creative aspect of their products. The brand, in other words, starts to represent the identity of a creative producer rather than a passive consumer. Often, companies will emphasize the ease of marketing and selling consumer-created products produced on their devices. Brands, though they still fulfill search cost reductions, take on a larger role, namely creating common identities. In this manner, OS communities, while sharing the common goal of open software, are subdivided into separate project communities, with operating system projects usually generating the most fervor and consistency towards participation.

An additional source of comparison is with non-profit organizations (NPOs). Indeed, some OS organizations are also NPOs, such as the Linux Foundation, Open Source Initiative (OSI 2008; formed by Eric Raymond and others), and the Mozilla Foundation. Like NPOs, OS communities are unlikely to maximize profits. OS projects also rely on volunteers, donations, and usually consists of smaller teams compared to firms. In a case study done by Gary et al. (2009), they found the OS company did not want to retire an earned brand name on the market, believing the project to be successful in the future and did not want the name brand associated with failure. OS projects, much like firms, have adopted slogans, mascots, logos, and other product differentiation methods in order to advertise and market their products. For instance, the logo of Linux is a penguin, for Firefox an orange fox, and for Debian a swirl representing open use. Oracle Linux[77] promotes its "unbreakable Linux" slogan. Branding alliances, particularly in the cases of IBM and OS, signals continual generation of innovation and an adherence to a goal and community. Indeed, for IBM, supporting OS helps drive interoperability, portability, and other features communities want.

As examined earlier, there is developer reputation within project communities. There is, additionally, the reputation of the entire community as a whole and their projects. Reputation is more often generated in written or video reviews and word of mouth on online forums. Either interested individuals, journalists, and developers of the project will create an overview of the upcoming changes (in animation, graphics, software packages, functions, etc.), comparing it to previous versions or related software. Discussion boards and blogs will do this as well.[78] Developers and their teams will have version release notes listing all the modifications. Quality assurance is therefore provided from external sources as well as internal.

B. Dispersal of Knowledge

Considering OS, Benkler (2002) wrote, "The widely distributed model of information production will better identify who is the best person to produce a specific component of a project, all abilities and availability to work on the specific module within a given time frame

[77] Formerly known as Oracle Enterprise Linux, is owned by the Oracle Corporation.

[78] Multiple tech news sites such as https://www.techradar.com/, https://techcrunch.com/, and https://gizmodo.com/; and blogs including https://medium.com/ and https://redmonk.com/ help inform users of the reputation and changes of each project.

considered." This is in stark contrast to Brooking's law (Brooks Jr., 1975/1995). Brooking's law is the notion that because an operating system—or any type of substantial software program—is immensely complicated, that would mean every programmer must know what the other programmer is doing in order to coordinate efforts and know where to contribute, thereby the coordination costs would be proportional to the square of the number of workers. This belief was the promonent paradigm in software development. Put simply, every incremental worker would increase the release date.[79] Therefore, software development must consist of a small, close-knit group of developers all following a structured plan. Moreover, Brooks Jr. continued, a top-down structure is necessary to delegate the specific complex tasks. A similar argument is made that as every economy becomes more global, increasing complexities arise and governments must intervene in order to direct and correct market failures.

In OS software development, there are reviewers who indeed make final decisions, but these reviewers do not dictate upon others specific goals or functions. Volunteers are free to submit any feature into the software, whenever they want. The reviewer simply draws from the well of knowledge provided by these voluntary submissions. Reviewers, in this manner, are much more in tune with user feedback than with proprietary development. Moreover, if volunteers remain unsatisfied since the code remains OS, they are free to copy or fork the project into their own version. There is, though, a taboo against forking, as new forks will divide the community and resources, features which otherwise would be implemented might not get properly integrated between platforms, and time spent developing on one fork is not spent on the other. Forking represents both a hazard and a benefit. In this way, OS software utilizes the dynamism in knowledge and preferences, allowing the system to be modified through different rules or combinations of stimuli (Hayek, 1967/1990b, p. 41).

There is no *a priori* method to know which system of rules will best fit. Each community must engage in a trial and error for different system of rules. Every member is affected by each rule created, for some the same rule affects them similarly and for others the same rule affects them differently. These rules, in turn, affect how the community develops and is governed, and also how new rules will be changed, removed, or implemented. In short, the community and the rules which govern it cannot be disentangled, much like

[79] Similar to the notion of the law of diminishing returns. The time and costs saved by adding an additional developer to a project would become outweighed by the increase in the time and costs of communicating, coordinating, and otherwise catching-up that developer.

political and private institutions (Wagner, 2016), though the two can be meaningfully but not completely examined and understood through ideal abstractions. Large companies have recognized the importance of dispersed knowledge and implemented programs to crowdsource[80] certain programs. Mozilla, Facebook, Yahoo, and Google have all began a "bug bounty program" where reports of bugs, exploits, and vulnerabilities can receive varying monetary compensation, depending on the severity of the issue. But due to the nature of proprietary software, the search for these bugs are still limited, as the underlying code is hidden. Commercial OS bounty programs have emerged (e.g. Bugcrowd,[81] Hackerone,[82] Open Bug Bounty,[83] VulnerabilityLab[84]) in response this problem. Another mechanism which aids in directing the attention of the community, though rare in OS, incorporates a system of community voting. One example is Netscape Communications' release of Bugzilla. Bugzilla is a free and OS bug tracking tool for various types in OS and proprietary software. Bugzilla is used in Mozilla, the Linux kernel, Apache, LibreOffice, and other projects. Anyone—users, observers, and developers—are able to vote on which bugs each of the projects should prioritize. As Dalle and den Besten (2010) found, bugs that are neglected or on the periphery, but remain important to the functioning of the project, consequently and consistently attract the most votes in Bugzilla. Another voting system was incorporated in the early development of Apache. Through e-mail, although any mailing list member could express an opinion, only the core developers' (usually less than ten members) votes mattered and were considered "binding." As Apache grew, more members gained the ability to vote, though accepted code required a "rough consensus" of 80-90 percent support (Deek & McHugh, 2007, p. 29).

Hippel and Krogh (2003) made a more general argument that the phenomenon of private benefits accruing from engaging in public goods is a form of "private-collective" model of innovative incentives. Moreover, by breaking software in smaller and smaller parts, and by dividing work up for more individuals, this allows for modular programming (Parnas, 1971). Parnas argued rather than to maximize communication between each module, the goal

[80] Crowdsourcing is not necessarily open but relies on individual and independent work. An individual or organization first presents a problem or challenge, the public attempts to solve the issue, and a platform is usually used to link the two together to complete the goal(s).

[81] https://www.bugcrowd.com/

[82] https://www.hackerone.com/

[83] https://www.openbugbounty.org/

[84] https://www.vulnerability-lab.com/

should be to minimize communication. Though, this minimized communication, explained in the earlier section Hacker Community and Culture, is not costless, taking time, effort, and social rapport to integrate into the OS community. In creating a system in which this is true, less time and effort is wasted through communication and the participants remain relatively autonomous. Coordination with low communication is made possible by modularity and granularity. Modularity is the number of modules, or distinct parts, within a project. Granularity is the size of the modules. The more people who get involved with a project, the smaller and more specialized each module becomes.[85] Above a certain threshold of modularity and granularity allows an individual to make the smallest investment in code production for OS development to become possible. Progress, technological and societal, therefore, is not a result of individuals becoming smarter or more productive but by breaking down tasks into more specific and diverse portions. Individuals, hence, are able to choose from the community, down to the project, and, if they so wished, contribute their efforts to a specific aspect or feature. Decomposability in complex systems, Simon (1962) highlighted, allows greater stability in an environment of uncertainty—more so in cases of Knightian uncertainty (1921/1964)—since a single piece can be altered, replaced, or removed without extinguishing the entire system. Firms and OS design computer systems not as a whole, but as parts into a larger system. Modularization makes complexity manageable, enables parallel work, and better accomodates future uncertainty (Baldwin & Hippel, 2011). In manufacturing, particularly the assembly line, modularity in production has been in use for over a century, like in car production.

Another advantage of modularized subcomponents is code reuse. Separate files and programs can be combined, or "linked," together into different configurations resulting in a different, larger program. In this fashion, programmers do not necessarily need to know how the entire project functions but can simply contribute to their primary specialty and focus. Volunteers are able to match the amount of time they which to contribute with the amount or size of each module. Computer scientists label this "distributed processing." Though more astute economists will recognize similar arguments related to Hayek's (1945) account of the price system as a dispersed, information-processing system. Of course, increasingly

[85] This phenomenon is related to the economic theorem: the division of labor is limited by the extent of the market (A. Smith, 1776/1998, ch. 3)

smaller module sizes cannot be taken to its logical conclusion, since each additional module would result in a proportionally decreasing progression and would require more and more individuals to complete—not to mention the increasing complexity in order to minimize the dependencies and putting every module together. In other words, there is then a trade-off between the size of the modules and how much progress should be made. Smaller modules decreases the daunting psychological barrier of completing a large, complicated task but at the same time increases the time and complexity of putting it all together into a whole. OS may be conceptualized as simply a microcosm of the larger political and social order, in which the supported claim from Hayek:

> [The orderliness of social activity] cannot be the result of a unified direction if we want individuals to adjust their actions to the particular circumstances largely known only to them and never known in their totality to any one mind. Order with reference to society thus means essentially that individual action is guided by successful foresight, that people not only make effective use of their knowledge but can also foresee with a high degree of confidence what collaboration they can expect from others (Hayek, 1960/2011, p. 229).

At the same time, multiple programmers may voluntarily work on the same task. This redundancy establishes continual overlap of knowledge and skills. If one programmer leaves the project for whatever reason, the modularity and redundancy allows the project to continue without collapsing or having to start all over again. This parallel development generates distributed intelligence that aids in the speed of bug fixing. In a Hayekian analysis, Garzarelli and Fontanella (2011) found that with some developers working on one task (horizontal division of labor) while others work on multiple tasks (vertical division of labor) allows for the individual to self-select, specialize, and simplify in correspondence to his or her primary expertise, while at the same time ferment creativity and discovery of new knowledge. V. Ostrom and E. Ostrom emphasized the same point,

> Duplication of functions is assumed to be wasteful and inefficient. Presumably efficiency can be increased by eliminating "duplication of services" and "overlapping jurisdictions." Yet we know that efficiency can be realized in a market economy only if multiple firms serve the same market. Overlapping service areas and

> duplicate facilities are necessary conditions for the maintenance of competition in a market economy (V. Ostrom & Ostrom, 1999, p. 94).

For a polycentrist, they would recognize two or more individuals—or communities—striving for the same goals would not be redundant, but fulfilling their personal goals and may lead to divergent solutions and knowledge. The community builds upon previous programming contributions and replaces any obsolete features in a spontaneous, destructive manner—in other words, creative destruction (Schumpeter & Swedberg, 1942/2003, p. 82-83).[86]

On the flip side, Langlois and Garzarelli (2008) argued trade-offs exist with modularization: the loss of integration and interchangeability. It may not be possible to fine tune and optimize a modular system compared to an integral one, though this depends on the system, and modular systems may certainly dominate a fine tuned one. The increased ability to recombine existing modules goes back to Adam Smith. He noted,

> Many improvements have been made by the ingenuity of the makers of the machines, when to make them became the business of a peculiar trade; and some by that of those who are called philosophers or men of speculation, whose trade it is not to do anything, but to observe everything; and who, upon that account, are often capable of combining together the powers of the most distant and dissimilar objects. (A. Smith, 1776/1998, book I, ch. I, p. 25)

Innovation, therefore, can inscribe a theory of recombination. Not all polycentric systems are necessarily efficient nor desirable. However, it is through polycentricity and competition in which the OS project and community that best satisfies needs in comparison to other projects and communities will grow and sustain.

C. Who Plans in an Open Source Environment?

During the socialist calculation and central planning debate,[87] Hayek emphasized without government intervention and regulation, there would still be planning, the question is who

[86] Though economist and professor at Chapman University Mark Skousen prefers the term "creative disruption" (Skousen, 2017, ch. 12).

[87] Though on each side individuals may not agree with one another nor fit into the definition, however roughly, in part, those on the central planning side were: Peter Kropotkin, Pierre-Joseph Proudhon, Cläre Tisch, Oskar Lange, and Abba Lerner. On the decentralized planning side: F.A. Hayek, Ludwig von Mises, Israel Kirzner, and Don Lavoie.

does the planning for whom (Hayek, 1945). It should be noted, however, that the concept of evolution was first described in the social sciences, which biology borrowed from.[88] Both Hayek and Polanyi used evolution, fusing it with the concepts of knowledge, cultures, and markets, in a similar way as biological evolution.[89] Hayek wrote "Cultural evolution, because it also rests on a sort of natural selection, looks very much like biological evolution. Both are likely to produce the results which Dr Darlington stresses, or to operate by 'hybridisation', 'recombination', 'assortative mating', or 'stratification'" (Hayek, 1978/1990a, p. 292). For Polanyi,

> The nature of scientific systems is more akin to the ordered arrangement of living cells which constitute a polycellular organism. The progress of science through the individual efforts of independent scientists is comparable in many ways to the growth of a higher organisim from a single microscopic germ-cell. Thoughout the process of embryonic development each cell pursues its own life, and yet each so adjusts its growth to that of its neighbours that a harmonious structure of the aggregate emerges. This is exactly how scientists co-operate: by continually adjusting their line of research to the results achived up to date by their fellow-scientists (Polanyi, 1951/1969, p. 35).

Though on the other hand, Mises (1981, p. 4) wrote, those who "attempt to conceive society as a biological organism" have "lost themselves in empty trivialities." Nonetheless, it appears sensible to assume that if human systems and biological systems are invariably and incomparably different, systematic human structure and knowledge which arises without central guidance on the surface is improbable, and even if possible, would produce impracticable results. But when we examine reality, it is clear from OS and other examples, this is not the case. There are natural laws which govern human behavior into finding complex solutions to complex problems. To best utilize the dispersed knowledge within society, the community, or other localities, well-defined orders or organizations is essential. For Hayek the rules and patterns in society can originate from an order or from an organization. Orders have rules that guide abstract behavior and are independent of purpose. Organizations have

[88] Hayek, 1973/2013, p. 22-23.

[89] The uses and benefits of biological analogies, specifically pertaining to the theory of the firm, has been debated against by Penrose (1952) and for by Alchian (1953).

rules that guide behavior with more or less concrete ends. Hayek builds upon Menger's distinction between "organic" and "pragmatic" institutions (Menger, 1883/1985), introducing spontaneous "orders" and planned "organizations" (Hayek, 1973/2013), see Table 4.1. Organic institutions are not created by the design of an individual or collective will, while pragmatic institutions are.

OS has evolved from unplanned and unintended decisions from thousands of actors, but incorporates an intended goal, placing it as an organic organization. Firms, in comparison, are pragmatic organizations. Langlois (1995) further explored the question of whether or not firms plan. OS, thus, operates in between the realm of our immediate friends and families and the extended order, continuing to "incorporate and generate knowledge which no individual brain, or any single organization, could possess or invent" (Hayek, 1988/1991, p. 72). Though in my earlier section that highlights community formation and the norms which arise from a seemingly constructionist point of view, these patterned behavior influencing developers are a product of past traditions, originating from other Internet communities or external societies, and for the most part, passed on tacitly. It is true though that these developers formalize and adjust some norms, in order to facilitate more effective cooperate behavior, but this is a small amount compared to the rest of knowledge and learned behaviors that have been followed from unjustified beliefs and shaped customs.

Table 4.1 Types of Orders and Organizations

	Orders	**Organizations**
Organic	Organic Orders Examples: language, common law, money, traffic	Organic Organizations Example: public choice view of government bureaucracies
Pragmatic	Pragmatic Orders Example: questionable if these exist[90]	Pragmatic Organizations Example: the firm and governments

(Source: Langlois, 1995)

[90] If these, in fact, do exist, see the works of constitutional economists like Buchanan and Wagner (1978) and Buchanan (1999).

It is also true, Hayek continued, the benefits and detriments of rules cannot be foreseen until adoption, and even then, remains dependent on the context of individuals and their circumstances. Any human interaction, online or offline, has along with it informal norms of conduct from leading to common, expected behaviors. Violations of these norms, through ignorance or purposeful action, can result in anger and termination of cooperation. Formal rules are rules that have been made explicit and codified, like licenses (e.g. GPL) and explicit agreements, also exist. Markets, firms, or OS communities with the least or most rules are not likely to facilitate continual cooperative behavior. It would make sense the more formalized rules that exist in an OS community, the more likely the community will become abandoned since it would constrict exploratory behavior, though this hypothesis requires more empirical research. The role of the managerial developers is not to deliberately arrange volunteers to do the necessary work but to induce conditions in which the volunteeers best arrange themselves. Both Hayek (1988/1991) and Polanyi (1951/1969) used the court of law and the legal community on the whole as an example of a polycentric order. V. Ostrom (1999, p. 62) noted upon this and wrote, "A fair judge is one who renders reasoned decisions that are considered to be reasonable by the various parties involved. A judge in a polycentric order is required to support his judgments both by the findings of fact and critical reasoning about the implications of legal relationships." It is not the sole judge who determines and imposes each ruling, but judges are part of an order which the communities have created and this community enforces what they recognize as a reasonable decision.

Furthermore, the appellate courts and the larger legal community all directly or indirectly weigh in on agreeing or scrutinizing the decision, organizing an evolving system of thought and legal structure. OS relies on the same mechanism. Leaders of OS communities recognize patterned behaviors and generate verbalized and written rules. They attempt to change these rules in order facilitate fairer accreditation of code, encourage more open cooperative behavior, or to reprimand bad behavior. Participants and other leaders, over a certain period of following these rules, will determine whether these changes are for the better and will discuss them, or leave (from OS altogether or to another community) in an attempt to provide feedback for these evolving rules. Other OS communities operates, like appellate courts and the overall legal community, by keeping in check bad rules and overzealous leaders. Though

every OS community has the same goal, that is, open code reuse and sharing, competition among themselves create overlap (in participants and) and fragmentation of authority. Explicit rules that achieve good outcomes in one OS community are not equally applicable to every community, and these leaders must use their intuition and experience to judge whether additional rules would be a benefit, and if these rules would be followed and enforced as intended.

D. Entrepreneurial Developers

OS development shrinks the gap between users and developers. Though the learning curve for using OS software is typically steep, particularly with some of the more advanced Linux operating systems which does not include pre-installed drivers or basic programs, like Arch Linux,[91] users have more information and community resources available to start engaging in light coding. Just as entrepreneurs[92] must remain alert to profit opportunities (Kirzner, 1997), entrepreneurial OS developers must remain alert to the desires and needs of the community. Entrepreneurship is simply a subset of all human action.[93] As such, the term "entrepreneur" in this paper is not confined to just start-ups, but entrepreneurial activity as an economic function of making judgmental decisions under uncertainty.[94] The first step

[91] Arch Linux is popularly known as one of the smallest and minimalist distributions of Linux. Installation is notoriously difficult for beginners, as installation requires command line prompts and additional setup for drivers requires knowledge of hardware information. Though Arch Linux provides one of the most functional and custom experiences possible for expert users.

[92] There are different theories of entrepreneurship, as Foss and Klein (2012, p. 23-42) compiled: (1) entrepreneurship as simple owner-managed firm (Knight, 1921/1964), (2) entrepreneurship as imagination or creativity, (3) entrepreneurship as innovation (Schumpeter, 1983, p. 128-156), entrepreneurship as alertness (Kirzner, 1999), entrepreneurship as adaptation (Schultz, 1980), entrepreneurship as a charismatic leader, and entrepreneurship as acting in uncertainty and using judgment (Mises, 1949/1998, p. 253-254). Though entrepreneurship as adaptation and entrepreneurship as a charismatic leader—some finding the charismatic leadership of Linus Torvalds essential (Bezroukov, 1999)–insightful, the Kirznerian framework remains the most applicable and explanatory in OS development, at least relative to this paper.

[93] "Living and acting man by necessity combines various functions. He is never merely a consumer. He is in addition either an entrepreneur, landowner, capitalist, or worker, or a person supported by the intake earned by such people. Problem-solving in markets, firms, or OS necessarily involves experimentation, not just a simple optimization problem. Moreover, the functions of the entrepreneur, the landowner, the capitalist, and the worker are very often combined by the same persons (Mises, 1949/1998, p. 253).

[94] It should be noted the difference here between risk and uncertainty. Risk is an event where the all possible outcomes are known and probabilities can be assigned to each outcome. For example, an unweighted six-sided dice (numbered one through six) has a one in six chance to roll numbers one through six. Uncertainty is an event where all possible outcomes are unknown or the probabilities are unknown for whatever reason. This is the case of most activities in human life. For example, opening a restaurant. Many factors like revenue, number of customers, how much food to order, etc., remains uncertain, though patterns may arise (such as more customers during weekends), outcomes remain unknown until the fact. In other words, there is no mathematical precision. Here, following the work of (Knight, 1921/1964), I emphasize the characteristic of uncertainty over risk as a more meaningful problem for OS and entrepreneurs in general.

is for developers who initiated OS projects to recognize there was some sort of inefficiency or deficiency with existing software and consequently the needs of users. Entrepreneurial developers are those who use the software, knows the community, and has tacit or explicit skills who then initiate an experiment to solve the problem they perceive. Since the needs and desires of consumers, the skills and interests of volunteers, and the world overall is everchanging and heterogeneous, entrepreneurs must make judgments in some part intuitively, that is, without perfect knowledge. Mokyr (2005), Ridley (2010), and Simpson (2013) argued most of the important inventions are not created by engineers or scientists, but are from the trial and error of the daily users, those who tinker and apply various techniques in order to increase work capacity that creates general and gradual progress. OS invites these tinkerers. Though advancements in machinery have progressed far beyond Adam Smith's imagination, these arguments still relate to his original observations on the improvements of what were in his day's simple manufacturing machinery.[95] The common man, therefore, is not only a laborer but is an inventor, making minute changes and innovations within his own productive efforts and capacities. OS software provides a more exploratory environment for users and developers tinker, thereby allowing them to experiment and stumble into better uses of the software. Hippel (2017) found millions of consumers around the world spend time and money to create and modify products which better suit their personal needs rather than programming to sell.

Entrepreneurial developers attempt to recombine existing resources and use the creativity provided by volunteers in order to convert low-use inputs to high-use outputs, though with limited reliance on the price mechanism. With limited pricing calculation and the profit and loss feedback, however, Knightian uncertainty is more pronounced. Initially, and for many OS projects, the founders did not attempt to interpret pricing data and arbitrage by turning lower priced goods into combinations of higher priced goods. With firms creating their own OS projects and certain OS communities becoming more firm-like, OS projects have engaged

[95] "All the improvements in machinery, however, have by no means been the inventions of those who had occasion to use the machines. Many improvements have been made by the ingenuity of the makers of the machines, when to make them became the business of a peculiar trade; and some by that of those who are called philosophers or men of speculation, whose trade it is not to do anything, but to observe everything; and who, upon that account, are often capable of combining together the powers of the most distant and dissimilar objects. In the progress of society, philosophy or speculation becomes, like every other employment, the principal or sole trade and occupation of a particular class of citizens" (A. Smith, 1776/1998, book I, ch. I, p. 25)

in Kirznerian interpretation of price data and also engaged in an act of creating a new good, which A. Martin (2010, p. 230) found to still exploit economic calculation. If developers, or human actors in general, do not know in advance the knowledge required competition will yield circumstances which lead to the discovery of relevant software. Developers who remain alert, use judgment, and implement the desired features will succeed in accumulating more users. In a case study done on the OS typesetting program LaTeX (typically stylized LaTeX), Gaudeul (2007) found developers who accepted suggestions made by non-developers, created a more user-friendly interface, and was able to reach a broader user-base. For OS software, rather than waiting for the developers to fix a problem, advanced users can patch their own software and send in the fix to the core developers, therefore the potentiality of new entries into the market remains high. Developers, in turn, are able to ask more directly to the users what they prefer. The fewer restrictions on the barrier to entry, the greater the competitive pressure, and the greater the incentive to create better products. Competition and trial and error fuel the discovery process (Hayek, 2002; Lachmann, 1988). Transaction costs have lowered enough that even without pricing, the mistakes developers and entrepreneurs make are not substantial enough to completely halt a project.

When projects fork, this creates an opportunity for entrepreneurial developers to come in and realize greater profits or efficiencies. As mentioned in the previous section on Open Source as Common-Pool Resource, there needs to be a balance struck between forking and standardization. Moreover, the problems of forking, that is, the dividing of time and resources, could result in either good or bad forks. There could be unmet coordination or inefficient resource usage. Developer managers who have a good sense of the comparative advantages of their employees and of various code-reuse from other projects will be more likely to succeed.

The open and free nature of OS software builds on the idea of "permissionless innovation" (Thierer, 2015), which allows for more tinkering than previous proprietary software.[96] The ability to think up new ideas and to experiment without precautionary blessings from a higher order—the boss, the bureaucrat, or the community—spurs technological progress

[96] Munger (2018, p. 15-17) found the same connection.

and inculcates the entrepreneurial spirit.[97] The result of enabling permissionless innovation creates greater choices, higher quality, and lowers costs (Thierer, 2015). If on the other hand, programmers had to convince the boss or the community what they were attempting was a good idea, they would have to rely on somebody else's knowledge, who may lack the insight or localized knowledge, to predict the success of the future endeavor. This would, of course, reduce the amount of trial and error experiments, which is foundational for OS software development. Moreover, the knowledge of experts and the mainstream, although useful in generalizing and recognizing certain patterns, have been argued to be narrowly applicable and results in confined success in markets or public policy (Easterly, 2013).[98] Programmers can simply start and submit their code. If those in charge like the submission, the code will be incorporated, if not—and the contributor programmer believes their submission is substantial and necessary—he or she is free to fork. The aim in OS is permissionless to submit; permissionless to fork. Innovation is by nature contrarian and will deviate from what is mainstream today. OS developers can only succeed only if they create and/or recombine products which benefits other users.

Langlois concluded firms are not advantageous over polycentric or noncephalic alternatives in all circumstances, but the advantage of "cephalization of deciding and controlling function" (Knight, 1921/1964, p. 268) in the firm is due to structural uncertainty in the environment. While many authors categorize OS software projects as networks (Grewal et al., 2006), others argued OS software organizations still contain some form of hierarchical structure (Scacchi, 2007). The list from the lowest ranking to the highest ranking members are: passive users and observers, active users, developers, project managers, community managers, and core developers (Scacchi, 2007). The difference between hierarchy in the firm and hierarchy in OS is that the hierarchy in OS is spontaneously organized. For firms, Williamson (1991, p. 271) wrote, "I maintain that hierarchy is not merely a contractual act but is also a contractual instrument, a continuation of market relations by other means."

The hierarchy maintained in OS is not through contracts, but as explored in previous parts of the paper, maintained mainly through intrinsic values, hacker norms, and community

[97] These ideas, limited by bounded rationality (Simon, 1955), are "cognitive maps" (Nelson, 2008) and provides a trajectory of what the project should achieve.

[98] For example, see the case of Fred Smith and FedEx by Munger (2017).

reputation. Furthermore, the hierarchy is created through a self-selecting process by various factors including each individual's motivation, time commitment, programming skill level, and interest. Benkler (2006, p. 105) noted "projects are based on a hierarchy of meritocratic respect, on social norms, and, to a great extent, on the mutual recognition by most players." Skill in programming is a necessary but not sufficient condition for leadership. The ability to lead and grow a community becomes increasingly important over time. Collectivism alone does not lead to detrimental results, though with coercive or forceful means, the worst do get on top (Hayek, 1944/2001, ch. 10). Voluntary collectivism (or in another term, freedom of association), with OS just being one of many examples, encourages the best to be on top. The different mechanisms for organization and coordination: markets/price, hierarchy/authority, and community/trust highlighted by Adler (2001), it would seem OS relies more heavily on the aspect of community/trust. Whereas the firm hierarchy determines who will collect the information, how to direct that information, and which decisions to form, OS displays all information freely and allows anyone the ability to direct and modify the information. Polanyi made the same argument for independently acting scientists and academic research:

> [A] scientist will select from the results obtained by others those elements which he can use best for his own task and will thus make the best possible contribution to science; opening thereby the field for other scientists to make their optimum contribution in their turn—and so on indefinitely (Polanyi, 1951/1969, p. 35).

In short, while the hierarchical structure may be just as essential in OS as in firms, the span of control, which was defined by Simon (1962) as the number of subordinates a single boss controls, is more limited, often directing fewer subordinates (and even here, in OS, the tasks a higher ranked member can assign to a lower ranked member is limited, if at all possible). In an OS hierarchical structure, work and deadlines are not assigned from a higher ranked member to a lower one. An OS hierarchy is a framework, like the legal framework, resting on community evaluations. Perhaps a more accurate claim is that OS does not rely on "managerial hierarchies" (Chandler, 1977) and can be seen as a mixture between hierarchy and the market (Ljungberg, 2000). The advantages of an authority in a hierarchy, Foss (2002) wrote, is not so easily replaceable. Firms—and this applies to OS projects as well—which hires and rewards employees which are able to ascend the rankings are able to (1) economize

on the transmittion of knowledge; rather than spending time explaining why an action is good, he or she can simply command it, (2) authority typically holds a wider picture and are able to direct the project in a clearer direction, and (3) authority is necessary to operate and maintain a reward systems.

Ioannides (2003) provided an excellent analysis of Hayekian spontaneous orders and organizations. Hayek explored three ways in which rules originate in society: (1) rules individuals obey because it is in accordance with their propensity towards such constraints or mental models, (2) rules individuals obey because they were followed by a common tradition, and (3) rules which are obeyed because of enforcement. At first, rules may not have been deliberately made, only followed, but people gradually consciously learned to improve these rules. Simple rules can create complex patterns of behavior. For Hayek, then, it is possible for spontaneous orders which rests on rules which are entirely the result of deliberate design (Hayek, 1973/2013, p. 44). OS experiments between different hierarchical structures, ranging from a more egalitarian approach (e.g. where anyone votes) to a rotating ruling group of contributors to a single decision maker. From firms to OS, this phenomenon continues "the gradual transformation of a rigidly organised hierarchic system into one where men could at least attempt to shape their own life" (Hayek, 1944/2001).[99] In any case, the organization attempts to balance condensing information effectively and making decisions well with freeing individuals to explore and experiment new features and fixes. The analysis of OS will benefit from a comparable analysis. The implementation phase consists of several sub-phases (Feller & Fitzgerald, 2002):

1. Code: writing code and submitting to the OS for review
2. Review: independent peer inspection of the code
3. Pre-commit test: breaking and integrating the submitted code in order to test the contribution carefully
4. Development release: rapid implementation of the submitted code—quick implementation helps motivate developers
5. Parallel debugging: large numbers of potential debuggers on various platforms and system configurations ensures bugs are found and fixed quickly
6. Production release: a relatively stable, debugged production version of the software is release for general users

This process, however, is more of a guideline and the management of the process varies

[99] Though Hayek was talking for the most part about despotic political and coercive hierarchic systems and firm hierarchies are almost incomparably less intrusive and remain contractual, there is still a continual trend in equality in markets and firms.

widely between different projects and communities.

Hippel (2017, p. 37-38) distinguished between three basic modes of innovation:

1. A *single free innovator* is an individual in the household sector which uses unpaid free time to develop and does not protect his or her design from free-riders.
2. A *collaborative free innovation project* involves unpaid household sector contributors who innovate and share their designs and does not protect it from free-riders.
3. A *producer innovator* is a single, non-collaborative firm. Using intellectual property rights to protect the design, they seek to retain exclusive control and to internalize as much of the benefits as possible.

The advantages of single free innovators are that they have no communication costs, if they choose not to actively advertise and attempt to diffuse their products to potential producers and adopters. The collaboration innovation project, Hippel notes, offers two major advantages: one, the output of the group is greater compared to separate individual innovators; and two, by working in a group, the range of innovative opportunities is greater since the project costs are greater than that of a single individual.[100]

The scalability of OS remains limited. From the analysis of Ohloh[101] data, Berkholz (2013) found for the largest OS projects, there were at most around 70 monthly unique committers and around 130 annual unique committers. In fact, merely one percent of all OS projects had 50 or more committers per year and 0.1 percent had 200 or more. This suggests an optimal amount of contributors in an OS community and project life, which is, in essence, a Dunbar's number (Dunbar, 1992) for OS community size. This makes sense, of course, since social ties and relations are more important in OS communities. Interestingly, Berkholz ignored the largest 100 projects, which had over 150 unique annual committers. These large projects are the most well known, such as GNOME, Chrome, KDE, the Linux kernel, and Mozilla, and would skew aggregate interpretation unrepresentative of general conclusions.

It seems once a projects reaches a high level of fame, community networking and individual rapport is no longer as important. Berkholz additionally noted upon the surprising feature of Pareto's principle,[102] stating 80 percent of the outcomes results from 20 percent of

[100] The advantages of a collaborative free innovation project are the same as the benefits of the division of labor (A. Smith, 1776/1998, book I, ch. I).

[101] Currently known as Black Duck Open Hub, or for short, Open Hub (see https://www.openhub.net/). Launched in 2006, the site looks to index projects, provide statistics about the longevity of the projects, and detail software metrics (lines of codes added, commit statistics, number of contributors, etc.).

[102] Also known as the 80/20 principle.

the causes. Microsoft also noted fixing 20 percent of its flawed code helps solve 80 percent of errors and crashes.[103] OS projects, like firms, exhibit tradeoffs when clustering resources. By arranging organization with greater amounts of trust, the social and transactional costs of continual contribution to OS projects is higher than in a firm or the market. As such, pure OS projects involve fewer contributors compared to what is possible in a firm. When projects become large and involve greater participants, there is more pressure to direct the project more in the traditional firm manner. Entrepreneurial OS developers, whether as a single innovator or within a collaborative project, are, then, still involved and plan through profit and loss by, as mentioned earlier in this paper, incorporating the price mechanism through bundling with proprietary software and relying on the pricing provided for by proprietary software. Another reason for smaller communities is that smaller groups are easier to keep track of reputation and to give credit for beneficial contributions. The satisfaction developers receive from seeing others benefit from his or her contributions cannot not be underestimated. Smaller groups, in general, consists of individuals who are more passionate and knowledgeable, like "special interest" groups and "vested interests" within the realm of politics. Smaller groups, then as Olson (1965/1971) and more recently, Bishin (2009) and A. Smith and Yandle (2014) argued, can certainly be more effective compared to large ones, and that small groups would be more likely, comparatively, in contributing to public goods.

V. The Case For Government Involvement

The role of state governments in OS can be separated into three parts:

1. Government as a funder
2. Government as a regulator
3. Government as a neutral arbiter

Or lastly, government can refrain from action and simply allow private governance, that is, let the developers and communities arrange the rules. As with government legislation, laws designed for one or multiple purpose(s) in mind may affect, positively or negatively, collaborations or communities in ways not intended by the drafters. This problem also occurs with community rules, though as explained earlier, there are feedback incentives

[103] See: Rooney (2002).

and institutional mechanisms which tend to prevent social problems from growing and perpetuating. The goal of this section is to provide arguments that other scholars have provided in order to better coordinate, protect, or encourage OS adoption through the means of government legislation, regulation, and/or funding. It will be up to the reader to decide whether private governance—explored earlier throughout this thesis—,government legislation, or a mix of both is to be the best solution for the continual growth of OS projects and communities. It is clear private organization has created a sufficient environment for entrepreneurial and reciprical developers to succeed, though there is always a possibility such environments could be further improved.

A. Government as a Regulator

The unintended consequences of technology and OS software have resulted in greater cooperation and community cohesion, forming consumer watchdogs, human-rights bodies, international trade opportunities, and much more. The effects, however, are not all positive. Consequently, dissemination of technology and its capabilities have also improved the abilities of nefarious criminals to steal money, identities, personal information, and engage in cyber espionage. The distance and scale criminals now can reach is unprecedented. After the terrorist attacks on September 11, 2001, a new area of vulnerability concern included cyberspace (Powell, 2005). Terrorist attacks and other harmful activities targeting the technological infrastructure and cyberspace may incorporate OS software and decentralized control of criminals. Automation in the workplace has improved productivity (also re-invoking fears job destruction), but criminals automation can now also set up automated and repeated attacks by uploading and re-uploading viruses and worms and committing distributed denial-of-service (DDoS) attacks. Governments have repeatedly been inept at addressing and protecting against cybercrimes, though this may be due to lack of political attention and state funding in this area. Some scholars have argued that governments are one of the main perpetrators of cybercrimes, calling for limitations on government monitoring, spying, and mass data collecting on its citizens.[104] The traditional methods of law enforcement reacting *ex-post* to crimes has proven, as a whole, inadequate.

Certain firms to solve this issue have partaken in a different strategy, although malignant

[104] Allen, 2008, 1, Farrell, 2014; Hayden, 2014.

criminal activities still exist. In the case of complex electronic commerce, the government could not address the small monetary amount but many separate occurrences in electronic theft. The company Paypal, however, was able to create *ex-ante*, rather than punishing criminals *ex-post*, solutions to fraud (Stringham, 2015, p. 100-115). That is, companies actively prevent problems and crimes before they arise. For OS, there have been research done on how to use OS organizational methods and software to assist in crime prevention. Jones (2007) likewise recommended *ex-ante* solutions for cybercrimes. He recognizes the proficiency of community-based policing in real neighborhoods and writes for a similar community policing in cybercrimes, specifically through OS software. The key to OS software success, Jones continues, is because development is distributed to the community; those who are best able to identify and correct potential security flaws are free to do so. Similarly, in the case of Linux and other OS projects, security vulnerabilities are much more quickly discovered than closed platforms, such as Windows.

Some scholars argue OS should be and has been adapted to monitor and predict crimes (Jones, 2007; Koops, 2013; Tompson, Johnson, Ashby, Perkins, & Edwards, 2015). Either through private or public means and enforcement remains contested. There has been little to no research done, however, the criminal uses of OS organization structures or its products. Certain operating systems, including Kali Linux, Parrot Security, and BlackArch, are created with tools of ethical hacking in mind, though, the tools could very well be used for nefarious reasons. Armed with penetration-testing tools like a password cracker, security scanners, and packet analyzers, these operating systems are dedicated to detect weaknesses in a computer and server system. Criminal use of OS would, then, seemingly provide a plausible and justifiable reason for government regulation and intervention. Similarly, proprietary software is sometimes forced to include backdoors and decryption methods in order to more easily access essential information of criminal activities. The problems that arise from the attempts of government to enforce technological crimes is the same as in the case of online fraud examined by Stringham (2005). Since technology, markets, and OS communities evolve quickly and often at in a localized and contextualized area, keeping up with the changes is extremely difficult. The labor costs, the sufficient availability of knowledgable government agents, and the incompatibility of different laws and legal procedures between countries

limits the ability of enforcement, according to Stringham. OS software in noncriminal use, explained throughout the previous parts of this paper, is extremely flexible and both users and developers exhibit rapid adoption and repudiation. This feature is what entices adoption in the first place. Criminals would use this feature as well.

Despite arguments for private governance in OS, tn the case of cybercrimes and espionage, M. Raymond (2016), noted on the success of government prohibitions on assassination, biological and chemical weapons, and certain classes of conventional weapons. On the cyber realm, he advised global governments to prohibit theft of intellectual property (including trade secrets or confidential business information) and government enabled cybercrimes (noting on Chinese hackers). Treaties and international law, Raymond continued, would only bind states. Since violations are commonplace and difficult to punish, he recommends implementing soft-law prohibition norms for small-scale crimes.

B. Government as a Funder

Financial support in any project is essential, which pays and incentivizes employees and volunteers to particpate in OS development, though, explained earlier, this factor is weaker compared to developers working in firms. OS academic research projects, at least in the United States, have been substantially funded through government agencies. For example, as Schweik and English (2012, p. 33) cited, before the term OS was even created, research contracts granted by the National Science Foundation and the Defense Advanced Research Projects Agency would require source code to be published. The BSD project was funded by the U.S. Department of Defense, contributing to the UNIX operating system. Government subsidized research grants allowed graduate students to develop all types of OS projects, including the hosting of public OS sites.

Government promotion of OS has been proposed and approved in many countries, most of them for research/development purposes or government administration, educational (schools and universities), and agency adoption of OS alternatives whenever viable. There have additionally been private endeavors to inspire young developers into OS projects. The prestigious program Summer of Code started by Google in 2005 has been steadily increasing, providing college students with exposure to distributed development, important ongoing OS

projects, and talented, more experienced developers in the field (Schweik & English, 2012, p. 24). Additional business motivations for promoting OS were highlighted in the section Symbiosis of Firms and Open Source. Though some legislation, particularly on state and local levels, promote OS more strongly.[105] For example, the government of Germany has struck a deal with IBM for them to provide hardware preinstalled with Linux provided by German distributor SuSE and Singapore has offered tax breaks to companies who use the Linux operating system instead of proprietary alternatives (United Nations Conference on Trade and Development, 2003, p. 116). Comino and Manenti (2005) and Comino, Manenti, and Rossi (2011) analyzed and found OS subsidies or mandating adoption in public agencies, schools, and universities may have positive public welfare benefits. As such, past efforts included Government Open Code Consortium and the annual Government Open Source Conference, and ongoing efforts include PloneGov and code.gov. Ghosh (2005) argued for stronger preferential treatment of OS. For Ghosh, governments should observe and regulate full competition for an open platform, even if a natural monopoly arises, and OS should be mandatory for all eGovernment software services.

C. Government as a Neutral Arbiter

The Internet, Lessig (1999a) explained, is governed through code, not through legislation. When setting up a server on the Internet, one must register and receive a name—a domain name—from a registry. In order to communicate with the server, users must transmit an address—the Internet Protocol (IP) address. This governance arose in necessity to keep track of and maintain order while allowing users to set up their websites and to visit other websites. The architecture of the Internet itself, Lessig (1999a) found, is governance through code. But this self-governance is not perfect. Lessig (1999b) later pointed out, the difference between physical regulatory spaces and cyberspaces take on different problems. Highlighting two examples, age in the real space is easily verifiable while in cyberspace, age is not self-authenticating. Laws which restrict on the basis of age, such as drinking and smoking, are much more difficult to certify age for cyberspace for adult content. Privacy, the second example, is easier to notice when infringed upon. Lessig noted while being tracked at a physical store, one would notice the cameras or an employee following you around. In cy-

[105] See https://www.csis.org/analysis/government-open-source-policies#sdendnote178sym

berspace, however, surveillance is difficult to reveal. Thus, Lessig recommended government regulation into either the code of cyberspace itself or to regulate the institutions (programmers, communities, volunteers, etc.) that produce the code. Though Lessig additionally noted, "My point is not to endorse such legislation: I think the ideal response for Congress is to do nothing. But if Congress adopted this form of regulation, my view is that it would be both feasible and constitution" (Lessig, 1999b, p. 518). Indeed, Lessig acknowledged other governance structures other than the state and through architecture: social norms and markets through prices.

For proprietary software, government regulation is easier. Because there is ownership of the code, governments can make laws and regulations directed at individuals or companies who own the code. Governments can influence both coordinating and regulating standards Lessig (1999c). Coordinating standards limit liberty while regulating standards limit the liberty within the activity. For example, driving on the right side of the road is a coordinating standard while the speed limit is a regulating standard. Yet for OS, both types of standards become much more complicated to control and enforce. For example, if a government wishes to regulate some illicit activity provided for by an OS software, and wishes to build its own version of the software, there would be no reason why users of the software would not simply substitute use for another software. The more open the code, (Lessig, 1999c) argued, the more difficult for government to regulate. However, the previous sections of this paper has shown users and contributors of OS create rules which tend to be more sustainable than possible externally imposed rules.

D. Intellectual Property: Public or Private Enforcement

Although OS remains in the realm of intellectual commons, there have been some issues with governmental regulation in terms of intellectual property and patents. The literature against intellectual property is numerous (Boldrin & Levine, 2002; Halbert, 2006; Shaffer, 2014; Zhou, 2011), though the literature supporting intellectual property is equally plentiful. Arguments against includes creating and maintaining unjust monopolies, economic inefficiencies, distortionary effects, and obstruction in the marketplace of ideas. The power of openness driving innovation is a belief OS contributors believe in. Supporting this belief, Wen,

Forman, and Graham (2013) found intellectual property enforcement for both business and personal use resulted in drastically decreased user interest and developer activity. Benkler (2002) argues strong intellectual property rights barricades existing information and raises the cost of accessing information. Despite these arguments, intellectual property rights remain contentious, in software and elsewhere.

Indeed, there exists strong hostility among some users against proprietary software and intellectual patents—to the point there they will only use OS software. On the other hand, there have been strong and pursusaive arguments for patents, intellectual property, and other forms of protection. Historically, as Machlup (1958) summarized, Smith,[106] Mill,[107] and Say, among other notable economists, favored patent protection, usually on ethical and pragmatic grounds—in the name of justice or natural rights and the promotion of public interest. Building upon similar sentiments, many scholars (Cockburn & MacGarvie, 2011; B. L. Smith & Mann, 2004; Weiser, 2003) believe the government was fundamental to the development of the Internet, the global position system (GPS), and other important technologies. Though, while OS emphatically states they are apolitical, Coleman (2004) argued the OS movement is rooted in the liberal traditions of free speech and individualism. Eisinger (1986), Mazzucato (2013), and Zhang (2016) made the case that government spurred on entrepreneurship and innovation in many industries, particularly the technological sector (for example, precursor to the Internet was the government created and funded ARPANET). To continue incentivizing innovation, the government must therefore set conditions for intellectual property, creating a "competitive platforms model" (Weiser, 2003). Before the 1976 Copyright Act, since software demand was still small developers negotiated individual license agreements with each purchaser. But as software costs became cheaper to copy and distribute, individual and separate contracts became untenable, and commercial software developers sought copyright coverage for computer programs (Heffan, 1997). Furthermore, Stiglitz (2008), while acknowledging the ambiguous nature of intellectual property and noting his ambivalence about the system, nonetheless argued for a system of intellectual property rights, though simplified. Additionally, he pointed out, monopolistic companies can be reigned in by forcing their patents out into the open, In other words, to deal with

[106] A. Smith, 1776/1998, book V, ch. I, pt. III.
[107] Mill, 1848/1909, book V, ch. X.

anticompetitive behavior, limiting a company's intellectual property rights is a useful tool. B. L. Smith and Mann (2004) also concluded the essential nature of intellectual property laws in technological innovation, noting second-comers and the "free-rider" problem. Copyright laws have, they argued, in effect, combated piracy, third-party appropriations, and allowed the engine of innovation while promoting broader goals of interoperability and transparency.

Menell (2011) noted intellectual property rights creates artificial scarcity in order to stimulate a market for an otherwise non-rivalrous and non-excludable good, arguing for a right to exclude. Indeed, one argument in favor for government is that the state has protected intellectual property rights of firms which allowed them to develop the technology enough to where OS is possible. Moreover, courts ultimately determine property rights and governments enforce the decision. The continuation of OS, it could be argued, should thus be guided by some, though not overbearing, state action. Further, Menell (2011) wrote, there must be a balance between incentives and property protection with antitrust law. By protecting intellectual property, firms can block competitors, improve goodwill reputation, and improve bargaining power on the market (Cohen, Nelson, & Walsh, 2000). Firms must make a decision whether formal (e.g. trademark, copyright, registered designs, or patent) or informal (e.g. secrecy, lead time, complexity, and confidentiality) intellectual protection is best, evaluating the effective enforcement regionally, the financial cost of filing for intellectual rights, and whether or not they can compete against other firms on pricing (Hall, Helmers, Rogers, & Sena, 2014). Most of these authors agree the current intellectual property rights regime is flawed in some way since no system can ever be perfect, but there is no need to massively overhaul or overthrow the entire concept, though this is exactly what the OS movement aims to do, at least regarding software programming.

VI. Conclusion

Many firms fail, many OS projects and communities unravel and fail. The concept of OS as a whole is not guaranteed success. Some proponents of OS software and FOSS overstate the importance of OS and too often ignores the foundation laid by proprietary software and the advantages provided for both developers and users. Both proprietary software and OS software have useful but also limiting features. As software development costs—

social and technological—continue to decrease, individuals have greater influence over the design and proliferation of the types of software they desire. The process of OS software development is remarkably insightful as to how complex and coordinated behavior arises without central planning or tightly enforced hierarchical structures. The phenomenon of common ownership in software is a result of many individual human actions but of no individual or group design. OS institutions which collect individuals together, which protects software from malicious intent, and which maintains updates and development are all a result of dispersed knowledge and the unintended consequences of human action. It is unlikely that OS will supplant markets or traditional firms. An interesting aspect undergirding most of Elinor Ostrom's is that the process of progress does not mean the path towards complete privatization nor the path towards complete openness; some amalgamation between the two is necessary for complex social cooperation. OS, like the Internet, is only just emerging, and the emergent properties of technology, particularly of the Internet in the past, were initially incorrectly evaluated and often underestimated. The ebb and flow between privatization and open property rights in cyberspace remains diverse (Benson, 2005). Though progress is gradual and slow, while entrepreneurs attempt by trial and error to find successful business and community models, the continued lower transaction costs and easier communication will make it increasingly easier to transition into a more open economy. For CPRs, there is still invariably a tradeoff between transaction cost and the increased social costs of maintaining a close-knit community, as in OS. The complexity of OS, and many spontaneous orders in general (although sponteneous orders need not be complex), is greater to any degree in which any single mind can ascertain or manipulate, and the continued success or the potential failure will offer more research in the area of collective action, polycentricity, and economics as a whole.

References

Abramitzky, R. (2018). *The Mystery of the Kibbutz: Egalitarian Principles in a Capitalist World*. The Princeton Economic History of the Western World. Princeton, New Jersey: Princeton University Press.

Adler, P. S. (2001). Market, Hierarchy, and Trust: The Knowledge Economy and the Future of Capitalism. *Organization Science*, *12*(2), 215–234.

Akerlof, G. A. (1970). The Market for "Lemons": Quality Uncertainty and the Market Mechanism. *The Quarterly Journal of Economics*, *84*(3), 488–500.

Alchian, A. A. (1953). Biological Analogies in the Theory of the Firm: Comment. *The American Economic Review*, *43*(4), 600–603.

Allen, A. L. (2008). The Virtuous Spy: Privacy as an Ethical Limit. *The Monist*, *91*, 3–22.

Apesteguia, J., & Maier-Rigaud, F. P. (2006, October). The Role of Rivalry: Public Goods Versus Common-Pool Resources. *Journal of Conflict Resolution*, *50*(5), 646–663.

Aristotle. (1998). *Politics* (C. D. C. Reeve, Trans.). Indianapolis, Cambridge: Hackett Publishing Company, Inc.

Arrow, K. J. (1962, June). The Economic Implications of Learning by Doing. *The Review of Economic Studies*, *29*(3), 155.

Baldwin, C. Y., & Hippel, E. von. (2011). Modeling a Paradigm Shift: From Producer Innovation to User and Open Collaborative Innovation. *Organization Science*, *22*(6), 1399–1417.

Baumol, W. J. (2004). *The Free-Market Innovation Machine: Analyzing the Growth Miracle of Capitalism*. Princeton, NJ: Princeton Univ. Press. (Original work published 2002)

Baumol, W. J., & Robyn, D. L. (2006). *Toward an Evolutionary Regime for Spectrum Governance: Licensing or Unrestricted Entry?* Washington, D.C: AEI-Brookings Joint Center for Regulatory Studies.

Benkler, Y. (2002, December). Coase's Penguin, or, Linux and "The Nature of the Firm". *The Yale Law Journal*, *112*(3), 369–446.

Benkler, Y. (2006). *The Wealth of Networks How Social Production Transforms Markets and Freedom*. New Haven; London: Yale University Press.

Benkler, Y., & Nissenbaum, H. (2006, December). Commons-Based Peer Production and Virtue. *Journal of Political Philosophy*, *14*(4), 394–419.

Benson, B. L. (2005). The Spontaneous Evolution of Cyber Law: Norms, Property Rights, Contracting, Dispute Resolution and Enforcement Without the State. *Journal of Law, Economics and Policy*, *1*(2), 269–348.

Berkholz, D. (2013, April 22). The Size of Open-Source Communities and Its Impact upon Activity, Licensing, and Hosting. *RedMonk*. Retrieved December 2, 2018, from https://redmonk.com/dberkholz/2013/04/22/the-size-of-open-source-communities-and-its-impact-upon-activity-licensing-and-hosting/

Bezroukov, N. (1999). Open Source Software Development as a Special Type of Academic Research (Critique of Vulgar Raymondism). *First Monday*, *4*(10). Retrieved from http://www.firstmonday.dk/ojs/index.php/fm/article/view/696/606

Bishin, B. G. (2009). *Tyranny of the Minority: The Subconstituency Politics Theory of Representation*. Philadelphia, PA: Temple University Press.

Block, W., & Barnett, W. (2009, July). Coase and Bertrand on Lighthouses. *Public Choice*, *140*(1-2), 1–13.

Boldrin, M., & Levine, D. (2002). The Case against Intellectual Property. *The American Economic Review*, *92*(2), 209–212.

Brooks Jr., F. P. (1995). *The Mythical Man Month: Essays on Software Engineering* (Anniversary Edition). USA: Addison Wesley Longman, Inc. (Original work published 1975)

Buchanan, J. M. (1965, February). An Economic Theory of Clubs. *Economica*, *32*(125), 1–14.

Buchanan, J. M. (1999). *The Logical Foundations of Constitutional Liberty*. Indianapolis: Liberty Fund, Inc.

Buchanan, J. M., & Wagner, R. E. (1978). *Fiscal Responsibility in Constitutional Democracy*. USA: Martinus Nijhoff Social Sciences Division.

Butters, G. R. (1976). A Survey of Advertising and Market Structure. *The American Economic Review*, *66*(2), 392–397.

Candela, R. A., & Geloso, V. J. (2018, September). The Lightship in Economics. *Public Choice*, *176*(3-4), 479–506.

Carver, B. W. (2005). Share and Share Alike: Understanding and Enforcing Open Source and Free Software Licenses. *Berkeley Technology Law Journal*, *20*(1), 443–481.

Chamberlin, E. H. (1950). Product Heterogeneity and Public Policy. *The American Economic Review*, *40*(2), 85–92.

Chamlee-Wright, E. (2010). *The Cultural and political Economy of Recovery: Social learning in a post-disaster environment*. Routledge Advances in Heterodox Economics. New York, NY: Routledge.

Chamlee-Wright, E., & Myers, J. A. (2008, September). Discovery and social learning in non-priced environments: An Austrian view of social network theory. *The Review of Austrian Economics*, *21*(2-3), 151–166.

Chandler, A. D. (1977). *The Visible Hand: The Managerial Revolution in American Business*. Cambridge, Mass: Belknap Press.

Coase, R. (1937). The Nature of the Firm. *Economica*, *4*(16), 386–405.

Coase, R. (1974). The Lighthouse in Economics. *Journal of Law and Economics*, *17*(2), 357–376.

Cockburn, I. M., & MacGarvie, M. J. (2011). Entry and Patenting in the Software Industry. *Management Science*, *57*(5), 915–933.

Cohen, W., Nelson, R., & Walsh, J. (2000, February). Protecting Their Intellectual Assets: Appropriability Conditions and Why U.S. Manufacturing Firms Patent (or Not). *National Bureau of Economic Research Working Paper 7552*.

Coleman, G. (2004). The Political Agnosticism of Free and Open Source Software and the Inadvertent Politics of Contrast. *Anthropological Quarterly*, *77*(3), 507–519.

Comino, S., & Manenti, F. M. (2005). Government Policies Supporting Open Source Software for the Mass Market. *Review of Industrial Organization*, *26*(2), 217–240.

Comino, S., Manenti, F. M., & Rossi, A. (2011, December 15). Public Intervention for Free/Open Source Software. *Revue d'économie industrielle*, (136), 89–108.

Conrad, K. (1982). Advertising, Quality and Informationally Consistent Prices. *Journal of Institutional and Theoretical Economics (JITE) / Zeitschrift für die gesamte Staatswissenschaft*, *138*(4), 680–694.

Cornes, R., & Sandler, T. (1994, July). Are Public Goods Myths? *Journal of Theoretical Politics*, *6*(3), 369–385.

Cowen, T. (1985). Public Goods Definitions and Their Institutional Context: A Critique of Public Goods Theory. *Review of Social Economy*, *43*(1), 53–63.

Crosetto, P. (2010). To Patent or Not to Patent: A Pilot Experiment on Incentives to Copyright in a Sequential Innovation Setting. In P. Ågerfalk, C. Boldyreff, J. M. González-Barahona, G. R. Madey, & J. Noll (Eds.), *Open Source Software: New Horizons* (Vol. 319, pp. 53–72). Berlin, Heidelberg: Springer Berlin Heidelberg.

Dalle, J.-M., & den Besten, M. (2010). Voting for Bugs in Firefox: A Voice for Mom and Dad? In P. Ågerfalk, C. Boldyreff, J. M. González-Barahona, G. R. Madey, & J. Noll (Eds.), *Open Source Software: New Horizons* (Vol. 319, pp. 73–84).

Deci, E. L. (1975). *Intrinsic Motivation*. Boston, MA: Springer US.

Deek, F. P., & McHugh, J. A. (2007). *Open Source: Technology and Policy*. Cambridge; New York: Cambridge University Press.

Demsetz, H. (1970, October). The Private Production of Public Goods. *The Journal of Law and Economics*, *13*(2), 293–306.

Deshpande, A., & Riehle, D. (2008). The Total Growth of Open Source. In B. Russo, E. Damiani, S. Hissam, B. Lundell, & G. Succi (Eds.), *Open Source Development, Communities and Quality* (Vol. 275, pp. 197–209).

Dorman, D. (2002). Open Source Software and the Intellectual Commons. *American Libraries*, *33*(11), 51–54.

Dunbar, R. (1992, June). Neocortex Size as a Constraint on Group Size in Primates. *Journal of Human Evolution*, *22*(6), 469–493

Easterly, W. (2013). *The Tyranny of Experts: Economists, Dictators, and the Forgotten Rights of the Poor*. New York: Basic Books, a member of the Perseus Book Group.

Eisinger, P. K. (1986). The Rise of the Entrepreneurial State In Economic Development. *Proceedings of the Annual Conference on Taxation Held under the Auspices of the National Tax Association-Tax Institute of America*, *79*, 34–44.

Engelhardt, S. (2011). What Economists Know About Open Source Software - Its Basic Principles and Research Results. *SSRN Electronic Journal*.

Eric Raymond on Hacking, Open Source, and the Cathedral and the Bazaar. (2009, January 19). Retrieved from http://www.econtalk.org/eric-raymond-on-hacking-open-source-and-the-cathedral-and-the-bazaar/

Farrell, H. (2014). Big Brother's Liberal Friends. *The National Interest*, (134), 25–34.

Feller, J., & Fitzgerald, B. (2002). *Understanding Open Source Development*. Boston, MA: Addison-Wesley Longman Publishing Co., Inc.

Fielding, R. T. (1999). Shared Leadership in the Apache Project. *Communications of the ACM*, *42*(4), 38–39.

Foss, N. J. (2002, February). 'Coase vs Hayek': Economic Organization and the Knowledge Economy. *International Journal of the Economics of Business*, *9*(1), 9–35.

Foss, N. J., & Klein, P. G. (2009). Austrian Economics and the Transaction Cost Approach to the Firm. *Libertarian Papers*, *1*(39), 1–20.

Foss, N. J., & Klein, P. G. (2012). *Organizing Entrepreneurial Judgment: A New Approach to the Firm*.

Frischmann, B. M., Madison, M. J., & Strandburg, K. J. (Eds.). (2014). *Governing Knowledge Commons*. USA: Oxford University Press.

Frischmann, B., Madison, M. J., & Strandburg, K. J. (2014). Governing Knowledge Commons. In B. M. Frischmann, M. J. Madison, & K. J. Strandburg (Eds.), *Governing Knowledge Commons* (pp. 1–43). USA: Oxford University Press.

Gary, K., Koehnemann, H., Blakley, J., Goar, C., Mann, H., & Kagan, A. (2009). A Case Study: Open Source Community and the Commercial Enterprise. In *2009 Sixth International Conference on Information Technology: New Generations* (pp. 940–945). 2009 Sixth International Conference on Information Technology: New Generations. Las Vegas, NV, USA: IEEE.

Garzarelli, G., & Fontanella, R. (2011). Open Source Software Production, Spontaneous Input, and Organizational Learning. *The American Journal of Economics and Sociology*, *70*(4), 928–950.

Gaudeul, A. (2007). Do Open Source Developers Respond to Competition? The (La)TeX Case Study.

Gaudeul, A. (2008). Open Source Licensing in Mixed Markets, or Why Open Source Software Does Not Succeed. *SSRN Electronic Journal*.

Ghosh, R. A. (2005). *An Economic Basis for Open Standards*. University of Maastricht Economic and social Research and training centre on Innovation and Technology (UNUMERIT); FLOSSPOLS project. Maastricht.

Goettsch, K. D. (2003). SCO Group v. IBM: The Future of Open-Source Software. *Journal of Law, Technology & Policy*, 581–588.

Greene, T. C. (2001, June 2). Ballmer: "Linux Is a Cancer". *The Register*. Retrieved from https://www.theregister.co.uk/2001/06/02/ballmer_linux_is_a_cancer/

Grewal, R., Lilien, G. L., & Mallapragada, G. (2006). Location, Location, Location: How Network Embeddedness Affects Project Success in Open Source Systems. *Management Science*, *52*(7), 1043–1056.

Haefliger, S., von Krogh, G., & Spaeth, S. (2008). Code Reuse in Open Source Software. *Management Science*, *54*(1), 180–193.

Halbert, D. J. (2006). *Resisting Intellectual Property*. New York: Taylor & Francis e-Library.

Halfaker, A., Geiger, R. S., Morgan, J. T., & Riedl, J. (2012). The Rise and Decline of an Open Collaboration System: How Wikipedia's Reaction to Popularity Is Causing Its Decline. *American Behavioral Scientist*, *57*(5), 664–688.

Hall, B., Helmers, C., Rogers, M., & Sena, V. (2014). The Choice between Formal and Informal Intellectual Property: A Review. *Journal of Economic Literature*, *52*(2), 375–423.

Hardin, G. (1968). The Tragedy of the Commons. *Science*, *162*(3859), 1243–1248.

Harris, C. (2018, October). Institutional Solutions to Free-Riding in Peer-to-Peer Networks: A Case Study of Online Pirate Communities. *Journal of Institutional Economics*, *14*(05), 901–924.

Hars, A., & Ou, S. (2001). Working for Free? – Motivations of Participating in Open Source Projects. In *Proceedings of the 34th Hawaii International Conference on System Sciences*. The Hawaii International Conference on System Sciences, Hawaii: IEEE.

Haruvy, E., Sethi, S. P., & Zhou, J. (2008, January). Open Source Development with a Commercial Complementary Product or Service. *Production and Operations Management*, *17*(1), 29–43.

Hayden, M. V. (2014). Beyond Snowden: An NSA Reality Check. *World Affairs*, *176*(5), 13–23.

Hayek, F. A. von. (1940, May). Socialist Calculation: The Competitive 'Solution'. *Economica*, *7*(26), 125.

Hayek, F. A. von. (1945). The Use of Knowledge in Society. *The American Economic Review*, *35*(4), 519–530.

Hayek, F. A. von. (1990a). Nature v. Nurture Once Again. In *New Studies in Philosophy, Politics, Economics and the History of Ideas* (Reprint, pp. 290–294). London: Routledge. (Original work published 1978)

Hayek, F. A. von. (1990b). *New Studies in Philosophy, Politics, Economics and the History of Ideas* (Reprint). London: Routledge. (Original work published 1967)

Hayek, F. A. von. (1991). *The Fatal Conceit* (Paperback edition) (W. W. Bartley III, Ed.). Chicago: The University of Chicago Press. (Original work published 1988)

Hayek, F. A. von. (1994). Foreword to the 1956 American Paperback Edition. In *The Road to Serfdom* (Fiftieth Anniversary Edition). USA: University of Chicago Press.

Hayek, F. A. von. (2001). *The Road to Serfdom*. Routledge Classics. London: Routledge. (Original work published 1944)

Hayek, F. A. von. (2002). Competition as a Discovery Procedure. *The Quarterly Journal of Austrian Economics*, *5*(3), 9–23.

Hayek, F. A. von. (2011). *The Constitution of Liberty* (Definitive Edition) (R. Hamowy, Ed.). The Collected Works of F. A. Hayek. Chicago: University of Chicago Press. (Original work published 1960)

Hayek, F. A. von. (2013). *Law, Legislation, and Liberty: A New Statement of the Liberal Principles of Justice and Political Econom*. London, New York: Routledge Classics. (Original work published 1973)

Heffan, I. V. (1997, July). Copyleft: Licensing Collaborative Works in the Digital Age. *Stanford Law Review*, *49*(6), 1487–1521.

Henkel, J. (2003). Welfare Implications of User Innovation. *MIT Sloan School of Management Working Paper No. 4237-03*.

Hippel, E. von. (1994). "Sticky Information" and the Locus of Problem Solving: Implications for Innovation. *Management Science*, *40*(4), 429–439.

Hippel, E. von. (2017). *Free Innovation*. Cambridge, MA: The MIT Press.

Hippel, E. von, & Krogh, G. von. (2003). Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science. *Organization Science*, *14*(2), 209–223.

Hirschman, A. O. (1970). *Exit, Voice, and Loyalty: Responses to Decline in Firms, Organizations, and States*. Cambridge, Mass: Harvard University Press.

Hoffman, E., Mccabe, K., & Smith, V. L. (2008). Chapter 46 Reciprocity in Ultimatum and Dictator Games: An Introduction. In C. Plott (Ed.), *Handbook of Experimental Economics Results* (Vol. 1). USA: North Holland.

Holcombe, R. G. (1997). A Theory of the Theory of Public Goods. *The Review of Austrian Economics*, *10*(1), 1–22.

Ioannides, S. (2003, July). Orders and Organizations: . Hayekian Insights for a Theory of Economic Organization. *American Journal of Economics and Sociology*, *62*(3), 533–566.

Jefferson, T. (1905). Letter from Thomas Jefferson to Isaac McPherson. In A. A. Lipscomb & A. E. Bergh (Eds.), *The Writings of Thomas Jefferson* (Vol. 20). Washington, D.C: Thomas Jefferson Memorial Association. (Original work published 1813)

Johnson, J. P. (2002, December). Open Source Software: Private Provision of a Public Good. *Journal of Economics & Management Strategy*, *11*(4), 637–662.

Johnson, R. N., & Libecap, G. D. (1982). Contracting Problems and Regulation: The Case of the Fishery. *The American Economic Review*, *72*(5), 1005–1022.

Jones, B. R. (2007). Comment: Virtual Neighborhood Watch: Open Source Software and Community Policing against Cybercrime. *The Journal of Criminal Law and Criminology*, *97*(2), 601–629.

Kapczynski, A., & Syed, T. (2013). The Continuum of Excludability and the Limits of Patents. *The Yale Law Journal*, *122*(7), 1900–1963.

Kealey, T. (1996). *The Economic Laws of Scientific Research*. Houndmills, Basingstoke, Hampshire, New York: Macmillan Press; St. Martin's Press.

Kirzner, I. M. (1997). Entrepreneurial Discovery and the Competitive Market Process: An Austrian Approach. *Journal of Economic Literature*, *35*(1), 60–85.

Kirzner, I. M. (1999). Creativity and/or Alertness: A Reconsideration of the Schumpeterian Entrepreneur. *The Review of Austrian Economics*, *11*, 5–17.

Klein, P. G. (1996). Economic Calculation and the Limits of Organization. *The Review of Austrian Economics*, *9*(2), 3–28.

Knight, F. H. (1964). *Risk, Uncertainty, and Profit* (Reprints of Economic Classics). New York: Augustus M. Kelley, Bookseller. (Original work published 1921)

Kogut, B. (2001, June 1). Open-Source Software Development and Distributed Innovation. *Oxford Review of Economic Policy*, *17*(2), 248–264.

Kollock, P. (1998). Design Principles for Online Communities. *PC Update*, *15*(5), 58–60.

Koops, B.-J. (2013, December). Police Investigations in Internet Open Sources: Procedural-Law Issues. *Computer Law & Security Review*, *29*(6), 654–665.

Krogh, G. von, Spaeth, S., & Lakhani, K. R. (2003, July). Community, Joining, and Specialization in Open Source Software Innovation: A Case Study. *Research Policy*, *32*(7), 1217–1241.

Kumar, V., Gordon, B. T., & Srinivasan, K. (2011). Competitive Strategy for Open Source Software. *Marketing Science*, *306*(2011), 1066–1078.

Kurrild-Klitgaard, P. (2010, June). Exit, Collective Action and Polycentric Political Systems. *Public Choice*, *143*(3-4), 339–352.

Lachmann, L. M. (1988). Speculative Markets and Economic Complexity. *Economic Affiars*, *8*(2), 7–10.

Lakhani, K., & Wolf, R. G. (2005). Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects. In J. Feller, B. Fitzgerald, S. Hissam, & K. R. Lakhani (Eds.), *Perspectives on Free and Open Source Software* (pp. 3–22). MIT Press.

Lange, O. (1936, October). On the Economic Theory of Socialism: Part One. *The Review of Economic Studies*, *4*(1), 53–71.

Langlois, R. N. (1995). Do Firms Plan. *Constitutional Political Economy*, *6*, 247–261.

Langlois, R. N. (2013, September). The Austrian Theory of the Firm: Retrospect and Prospect. *The Review of Austrian Economics*, *26*(3), 247–258.

Langlois, R. N., & Garzarelli, G. (2008, April). Of Hackers and Hairdressers: Modularity and the Organizational Economics of Open-source Collaboration. *Industry and Innovation*, *15*(2), 125–143.

Laurent, C. (2003). The Political Economy of Lighthouses. *Public Choice*, *157*(1), 51–56.

Lea, G. (2000, July 31). MS' Ballmer: Linux Is Communism. *The Register*. Retrieved from https://www.theregister.co.uk/2000/07/31/ms_ballmer_linux_is_communism/

Leeson, P. T., & Skarbek, D. B. (2010). Criminal Constitutions. *Global Crime*, *11*(3), 279–297.

Lerner, A. P. (1934, October). Economic Theory and Socialist Economy. *The Review of Economic Studies*, *2*(1), 51–61.

Lerner, J., Pathak, P. A., & Tirole, J. (2006). The Dynamics of Open-Source Contributors. *The American Economic Review*, *96*(2), 114–118.

Lerner, J., & Tirole, J. (2002). Some Simple Economics of Open Source. *The Journal of Industrial Economics*, *50*(2), 197–234.

Lerner, J., & Tirole, J. (2005). The Economics of Technology Sharing: Open Source and Beyond. *The Journal of Economic Perspectives*, *19*(2), 99–120.

Lessig, L. (1999a). Open Code and Open Societies: Values of Internet Governance. *Chicago-Kent Law Review*, *74*(3), 1405–1420.

Lessig, L. (1999b, December). The Law of the Horse: What Cyberlaw Might Teach. *Harvard Law Review*, *113*(2), 501–549.

Lessig, L. (1999c). The Limits in Open Code: Regulatory Standards and the Future of the Net. *Berkeley Technology Law Journal*, *14*(2), 759–769.

Lessig, L. (2008). *Remix: Making Art and Commerce Thrive in the Hybrid Economy*. USA: The Penguin Press.

Levy, S. (2010). *Hackers: Heroes of the Computer Revolution* (1st ed). Sebastopol, CA: O'Reilly Media. (Original work published 1984)

Ljungberg, J. (2000, December). Open Source Movements as a Model for Organising. *European Journal of Information Systems*, *9*(4), 208–216.

Machlup, F. (1958). An Economic Review of the Patent System (1958). *Study No. 15, US. Senate Committee on the Judiciary, Subcommittee on Patents, Trademarks and Copyrights*.

Martin, A. (2010). Emergent Politics and the Power of Ideas*. *Studies in Emergent Order*, *3*, 212–245.

Martin, N. P., & Storr, V. H. (2008). On Perverse Emergent Orders: *Studies in Emergent Order*, *1*, 73–91.

Maurer, S., & Scotchmer, S. (2006, April). Open Source Software: The New Intellectual Property Paradigm. In *Handbook of Economics and Information Systems* (pp. 285–319).

Mauss, M. (2002). *The Gift: The Form and Reason for Exchange in Archaic Societies*. Routledge Classics. London: Routledge. (Original work published 1950)

Mazzucato, M. (2013). *The Entrepreneurial State: Debunking Public vs Private Myths in Risk and Innovation*. New York, London: Anthem Press.

McAllister, D. J. (1995). Affect- and Cognition-Based Trust as Foundations for Interpersonal Cooperation in Organizations. *The Academy of Management Journal*, *38*(1), 24–59.

McCulloch, J. R. (2013). *The Principles of Political Economy*. Indianapolis: Liberty Fund, Inc. (Original work published 1864)

McGee, K., & Skågeby, J. (2004). Gifting Technologies. *First Monday*, *9*(12). Retrieved from https://firstmonday.org/ojs/index.php/fm/article/view/1192/1112

Mehra, A., Dewan, R., & Freimer, M. (2011). Firms as Incubators of Open-Source Software. *Information Systems Research*, *22*(1), 22–38.

Menell, P. S. (2011). Governance of Intellectual Resources and Disintegration of Intellectual Property in the Digital Age. *Berkeley Technology Law Journal*, *26*(4), 1523–1559.

Menger, C. (1985). *Investigations into the Method of the Social Sciences with Special Reference to Economics.* (L. Schneider, Ed. & F. J. Nock, Trans.). New York, London: New York University Press. (Original work published 1883)

Mill, J. S. (1909). *Principles of Political Economy with Some of Their Applications to Social Philosophy* (7th) (S. J. Ashley, Ed.). London: Longmans, Green and Co. (Original work published 1848)

Mill, J. S. (2001). *On Liberty*. Kitchener, Ontario: Batoche Books Limited. (Original work published 1859)

Mises, L. von. (1962). *Socialism: An Economics and Sociological Analysis* (New Edition) (J. Kahane, Trans.). New Haven: Yale University Press. (Original work published 1951)

Mises, L. von. (1981). *Epistemological Problems of Economics* (G. Reisman, Trans.). USA: New York University Press.

Mises, L. von. (1998). *Human Action: A Treatise on Economics* (Scholar's ed.). Auburn, Alabama: Ludwig von Mises Institute. (Original work published 1949)

Mises, L. von. (2007). *Theory and History: An Interpretation of Social and Economic Evolution*. Auburn, Alabama: Ludwig von Mises Institute. (Original work published 1957)

Mises, L. von. (2008). *Profit and Loss*. Auburn, Alabama: Ludwig von Mises Institute.

Mises, L. von. (2012). *Economic Calculation in the Socialist Commonwealth*. Auburn, Alabama: Ludwig von Mises Institute. (Original work published 1920)

Moglen, E. (1999). Anarchism Triumphant: Free Software and the Death of Copyright. *First Monday*, *4*(8). Retrieved from http://firstmonday.org/ojs/index.php/fm/article/view/684/594

Mokyr, J. (2005). *The Gifts of Athena: Historical Origins of the Knowledge Economy*. Princeton, NJ: Princeton Univ. Press.

Munger, M. (2017, September 19). Permissionless Innovation: The Fuzzy Idea That Rules Our Lives. *Learn Liberty*. Retrieved October 26, 2018, from http://www.learnliberty.org/blog/permissionless-innovation-the-fuzzy-idea-that-rules-our-lives/

Munger, M. (2018). *Tomorrow 3.0_ Transacting Costs and the Sharing Economy*. New York: Cambridge University Press.

Nelson, R. R. (1989, July). What Is Private and What Is Public About Technology? *Science, Technology, & Human Values*, *14*(3), 229–241.

Nelson, R. R. (2008, July). Bounded Rationality, Cognitive Maps, and Trial and Error Learning. *Journal of Economic Behavior & Organization*, *67*(1), 78–89.

Nyman, L., & Mikkonen, T. (2011). To Fork or Not to Fork: Fork Motivations in SourceForge Projects. *International Journal of Open Source Software and Processes*, *3*(3).

O'Mahony, S. (2003, July). Guarding the Commons: How Community Managed Software Projects Protect Their Work. *Research Policy*, *32*(7), 1179–1198.

Olson, M. (1971). *The Logic of Collective Action: Public Goods and the Theory of Groups*. Harvard Economic Studies. Cambridge, MA; London, England: Harvard University Press. (Original work published 1965)

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

Ostrom, E. (1999). Polycentricity, Complexity, and the Commons. *The Good Society*, *9*(2), 37–41.

Ostrom, E. (2000). Collective Action and the Evolution of Social Norms. *Journal of Economic Perspectives*, *14*(3), 137–158.

Ostrom, E. (2005). *Understanding Institutional Diversity*. Princeton, NJ: Princeton Univ. Press.

Ostrom, E., Gardner, R., & Walker, J. (1994). *Rules, games, and common-pool resources*. Ann Arbor: University of Michigan Press.

Ostrom, E., & Hess, C. (2007a). A Framework for Analyzing the Knowledge Common. In E. Ostrom & C. Hess (Eds.), *Understanding Knowledge as a Commons: From Theory to Practice* (pp. 41–81). Cambridge, MA; London, England: The MIT Press.

Ostrom, E., & Hess, C. (Eds.). (2007b). *Understanding Knowledge as a Commons: From Theory to Practice*. Cambridge, MA; London, England: The MIT Press.

Ostrom, V. (1999). Polycentricity (Part 1). In M. D. McGinnis (Ed.), *Polycentricity and Local Public Economies* (pp. 52–74). USA: University of Michigan Press.

Ostrom, V., & Ostrom, E. (1999). Public Goods and Public Choices. In M. D. McGinnis (Ed.), *Polycentricity and Local Public Economies* (pp. 75–103). USA: University of Michigan Press.

Park, Y., & Jensen, C. (2009, September). Beyond Pretty Pictures: Examining the Benefits of Code Visualization for Open Source Newcomers. In *2009 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis* (pp. 3–10). 2009 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT).

Parks, R. B., Baker, P. C., Kiser, L. L., Oakerson, R. J., Ostrom, E., Ostrom, V., … Wilson, R. K. (1999). Consumers as Coproducers of Public Services: Some Economic and Institutional Considerations. In M. D. McGinnis (Ed.), *Polycentricity and Local Public Economies* (pp. 381–391). USA: University of Michigan Press.

Parnas, D. L. (1971). On the Criteria to Be Used in Decomposing Systems into Modules. *Carnegie-Mellon University*.

Penrose, E. (1952). Biological Analogies in the Theory of the Firm. *The American Economic Review*, *42*(5), 804–819.

Pigou, A. C. (1920). *The Economics of Welfare*. Macmillan and Co., Ltd.

Pitt, L. F. (2006, April 1). The Penguin's Window: Corporate Brands From an Open-Source Perspective. *Journal of the Academy of Marketing Science*, *34*(2), 115–127.

Polanyi, M. (1969). *The Logic of Liberty: Reflections and Rejoinders* (Fourth Impression). USA: University of Chicago Press. (Original work published 1951)

Potts, J. (2012). Novelty-Bundling Markets. In *The Spacial Market Process (Advances in Austrian Economics, Volume 16)* (pp. 291–312). Emerald Group Publishing Limited.

Powell, B. (2005). Is Cybersecurity a Public Good? Evidence From The Financial Services Industry. *Journal of Law, Economics and Policy*, *1*(2), 497–510.

Raymond, E. (2002). *The Cathedral and the Bazaar: Musings on Linux and Open Bource By an Accidental Revolutionary* (Revised). USA: O'Reilly & Associates, Inc. (Original work published 1999)

Raymond, M. (2016). Managing Decentralized Cyber Governance The Responsibility to Troubleshoot. *Strategic Studies Quarterly*, *10*(4), 123–149.

Ridley, M. (2010). *The Rational Optimist: How Prosperity Evolves*. USA: Harper Perennial.

Roa, A. R., & Ruekert, R. W. (1994). Brand Alliances as Signals of Product Quality. *Sloan Management Review*, 87–97.

Romer, P. M. (1990). Endogenous Technological Change. *The Journal of Political Economy*, *98*(5), S71–S102.

Rooney, P. (2002, October). Microsoft's CEO: 80-20 Rule Applies To Bugs, Not Just Features. *CRN*. Retrieved January 17, 2019, from https://www.crn.com/news/security/18821726/microsofts-ceo-80-20-rule-applies-to-bugs-not-just-features.htm

Rosenberg, N. (1976). *Perspectives on Technology*. Cambridge; New York: Cambridge University Press.

Rothbard, M. N. (2004). *Man, Economy, and State with Power and Market: Government and Economy* (Second Edition, Scholar's Edition). Auburn, Alabama: Ludwig von Mises Institute. (Original work published 1962)

Rousseau, J.-J. (2002). *The Social Contract and The First and Second Discourses* (S. Dunn, Ed.). New Haven; London: Yale University Press.

Salerno, J. T. (1995). A Final Word: Calculation, Knowledge, and Appraisement. *The Review of Austrian Economics*, *9*(1), 141–142.

Samuelson, P. A. (1954). The Pure Theory of Public Expenditure. *Review of Economics and Statistics*, *36*(4), 397–89.

Sauer, R. M. (2007). Why Develop Open-Source Software? The Role of Non-Pecuniary Benefits, Monetary Rewards, and Open-Source Licence Type. *Oxford Review of Economic Policy*, *23*(4), 605–619.

Scacchi, W. (2007). Free/Open Source Software Development: Recent Research Results and Emerging Opportunities. In *ESEC-FSE Companion '07 The 6th Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering: Companion Papers* (pp. 459–468). Dubrovnik, Croatia: ESEC-FSE.

Schoeck, H. (1987). *Envy: A Theory of Social Behavior*. USA: Liberty Fund, Inc. (Original work published 1966)

Schultz, T. W. (1980). Investment in Entrepreneurial Ability. *The Scandinavian Journal of Economics*, *82*(4), 437–448.

Schumpeter, J. A. (1983). *Theory of Economic Development: An Inquiry into Profits, Capital, Credit, Interest, and the Business Cycle* (R. Opie, Trans.). New Brunswick, London: Transaction Publishers.

Schumpeter, J. A., & Swedberg, R. (2003). *Capitalism, Socialism, and Democracy*. USA: Taylor & Francis e-Library. (Original work published 1942)

Schwartz, A., & Wilde, L. L. (1983). Imperfect Information in Markets for Contract Terms: The Examples of Warranties and Security Interests. *Virginia Law Review*, *69*(8), 1387–1485.

Schwarz, M., & Takhteyev, Y. (2010, July 19). Half a Century of Public Software Institutions: Open Source as a Solution to Hold-Up Problem. *Journal of Public Economic Theory*, *12*(4), 609–639.

Schweik, C. M., & English, R. C. (2012). *Internet Success: A Study of Open-Source Software Commons*. Cambridge, Mass: MIT Press.

Sen, R., Subramaniam, C., & Nelson, M. L. (2008, December). Determinants of the Choice of Open Source Software License. *Journal of Management Information Systems*, *25*(3), 207–240.

Shaffer, B. (2014). *Libertarian Critique Of Intellectual Property, A*. Auburn, Alabama: Ludwig von Mises Institute.

Shaw, J. S. (2010). Education—A Bad Public Good? *The Independent Review*, *15*(2), 241–256.

Sidgwick, H. (1887). *The Principles of Political Economy* (Second). London: Macmillan and Co., Ltd.

Simon, H. A. (1955, February). A Behavioral Model of Rational Choice. *The Quarterly Journal of Economics*, *69*(1), 99–118.

Simon, H. A. (1962). The Architecture of Complexity. *Proceedings of the American Philosophical Society*, *106*(6), 467–482.

Simpson, D. (2013). *The Rediscovery of Classical Economics: Adaptation, Complexity and Growth*. Cheltenham, UK; Northampton, MA: Edward Elgar.

Skousen, M. (2017). *Economic Logic* (Fifth Edition). USA: Simon and Schuster.

Smith, A. (1998). *An Inquiry Into the Nature and Causes of the Wealth of Nations*. London: The Electric Book Company Ltd. (Original work published 1776)

Smith, A., & Yandle, B. (2014). *Bootleggers & Baptists: How Economic Forces and Moral Persuasion Interact to Shape Regulatory Politics*. USA: Cato Institute.

Smith, B. L., & Mann, S. O. (2004). Innovation and Intellectual Property Protection in the Software Industry: An Emerging Role for Patents? *The University of Chicago Law Review*, *71*(1), 241–264.

Snidal, D. (1979, December). Public Goods, Property Rights, and Political Organizations. *International Studies Quarterly*, *23*(4), 532–566.

Steinmacher, I., Conte, T., Gerosa, M. A., & Redmiles, D. (2015). Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing - CSCW '15* (pp. 1379–1392). The 18th ACM Conference.

Stewart, & Gosain. (2006). The Impact of Ideology on Effectiveness in Open Source Software Development Teams. *MIS Quarterly*, *30*(2), 291.

Stigler, G. E. (1961). The Economics of Information. *The Journal of Political Economy*, *69*(3), 213–225.

Stiglitz, J. E. (2008). Economic Foundations of Intellectual Property Rights. *Duke Law Journal*, *57*(6), 1693–1724.

Strahilevitz, L. (2003). Charismatic Code, Social Norms, and the Emergence of Cooperation on the File-Sharing Networks. *Virginia Law Review*, *89*(3), 505–595.

Stringham, E. P. (2005). The Capability of Government in Providing Protection Against Online Fraud: Are Classical Liberals Guilty of the Nirvana Fallacy? *Journal of Law, Economics and Policy*, *1*(2).

Stringham, E. P. (2015). *Private Governance: Creating Order in Economic and Social Life*. New York: Oxford University Press.

Taylor, J. (2001, September). Private Property, Public Interest, and the Role of the State in Nineteenth-Century. *The Historical Journal*, *44*(3), 749–771.

Thierer, A. D. (2015). The Internet of Things and Wearable Technology: Addressing Privacy and Security Concerns without Derailing Innovation. *Richmond Journal of Law and Technology*, *21*(2).

Thomas, B. K. (2010). Participation in the Knowledge Society: The Free and Open Source Software (FOSS) Movement Compared with Participatory Development. *Development in Practice*, *20*(2), 270–276.

Thomas, D. (2002). *Hacker Culture*. Minneapolis: University of Minnesota Press.

Tiebout, C. M. (1956). A Pure Theory of Local Expenditures. *Journal of Political Economy*, *64*(5), 416–424.

Tompson, L., Johnson, S., Ashby, M., Perkins, C., & Edwards, P. (2015, March 15). UK Open Source Crime Data: Accuracy and Possibilities for Research. *Cartography and Geographic Information Science*, *42*(2), 97–111.

Tsai, B. J. (2008). For Better or Worse: Introducing the GNU General Public License Version 3. *Berkeley Technology Law Journal*, *23*(1), 547–581.

United Nations Conference on Trade and Development. (2003). *E-Commerce and Development Report 2003*. New York: United Nations.

Varughese, G., & Ostrom, E. (2001, May). The Contested Role of Heterogeneity in Collective Action: Some Evidence from Community Forestry in Nepal. *World Development*, *29*(5), 747–765.

Wagner, R. (2016). *Politics as a Peculiar Business: Insights from a Theory of Entangled Political Economy*. New Thinking in Political Economy. Cheltenham, UK; Northhampton, MA: Edward Elgar Publishing.

Weber, S. (2004). *The Success of Open Source*. Cambridge, MA: Harvard University Press.

Weiser, P. J. (2003, April). The Internet, Innovation, and Intellectual Property Policy. *Columbia Law Review*, *103*(3), 534–613.

Wen, W., Forman, C., & Graham, S. J. H. (2013). Research Note: The Impact of Intellectual Property Rights Enforcement on Open Source Software Project Success. *Information Systems Research*, *24*(4), 1131–1146.

Williams, M. R., & Hall, J. C. (2015, December). Hackerspaces: A Case Study in the Creation and Management of a Common Pool Resource. *Journal of Institutional Economics*, *11*(04), 769–781.

Williamson, O. E. (1985). *The Economic Institutions of Capitalism*. USA: The Free Press.

Williamson, O. E. (1991, June). Comparative Economic Organization: The Analysis of Discrete Structural Alternatives. *Administrative Science Quarterly*, *36*(2), 269–296.

Zandt, D. E. van. (1993). The Lessons of the Lighthouse: "Government" or "Private" Provision of Goods. *The Journal of Legal Studies*, *22*(1), 47–72.

Zhang, H. (2016). *China's Local Entrepreneurial State and New Urban Spaces*. New York: Palgrave Macmillan US.

Zhou, Y. (2011). Against Intellectual Monopoly: Free Software in China. *World Review of Political Economy*, *2*(2), 290–306.

Zhu, K., & MacQuarrie, B. (2003, September 1). The Economics of Digital Bundling: The Impact of Digitization and Bundling on the Music Industry. *Communications of the ACM*, *46*(9), 264–270.