HOW CAN OPEN SOURCE AND CLOSED SOURCE SOFTWARE BUSINESS STRUCTURES MUTUALLY EXIST

Yugank Goyal
Padmanabha Ramanujam

Available at: https://works.bepress.com/yugank_goyal/1/
HOW CAN OPEN SOURCE AND CLOSED SOURCE SOFTWARE BUSINESS STRUCTURES MUTUALLY EXIST

Ramanujam Padmanabha and Yugank Goyal*

ABSTRACT

The phenomenon of producing Open Source Software based on unconstrained access to source code and the swift growth of the open source business structure of producing software fuelled by Linux Operating System and Apache Web Server have raised important questions, which are of immense scholastic interest. Accordingly many scholars in the last few years have endeavoured to clarify as to why thousands of top-quality programmers contribute freely to an open source product which is a public good. However, there has hardly been any attempt to explain how open source and closed source business structure of producing software can coexist. This paper is an effort in this direction. The paper presents two distinct game theoretic models to explain why open source programmers will coexist alongside closed source programmers.

Key Words: Open Source, Signalling Incentives, Motivation

JEL classification: L31, H40, C70

*Authors are Assistant Professors at Jindal Global Law School and Assistant Directors of Centre for IPR Studies at O.P. Jindal Global University, India. They are grateful to Prof. Thomas Eger, Institute of Law and Economics, University of Hamburg, Germany and also Prof. Dr. Ramakrishna, National Law School of India University, India for providing useful feedback and comments. Research Assistance of Mr. Gaurav Sharma, student, Jindal Global Law School, is also appreciated. Any errors or omissions are purely authors’. Authors can be reached at pramanujam@jgu.edu.in and ygoyal@jgu.edu.in
# TABLE OF CONTENTS

A. Introduction .................................................. 3  
B. Open Source Advocates v/s Closed Source Proponents ....... 6  
C. What Motivates the Open Source Programmer to Contribute Software for Free ........ 9  
D. Open Source and Closed Source Business Structure can mutually exist ................. 15  
  a. Whether co-existence of open source and closed source programmers is an evolutionarily stable strategy .... 20  
E. Conclusion ................................................... 22
INTRODUCTION

When Thomas Kuhn\(^1\) published his seminal work in 1962 (book entitled, *The Structure of Scientific Revolutions*), he argued that scientific progress is marked by epochal change and is not a result of oft-thought gradual progress. Whether it was Newton to explain the laws of gravity or Darwin’s theory of origin of species, none of the path-breaking discoveries were built on existing discoveries.\(^2\) These revolutionary processes were called by him, *paradigm shifts*.

A brilliant – and relevant to our discussion – paradigm shift took place in 1981 when IBM introduced the standard architecture of personal computers, as opposed to the contemporary industry practice. The new paradigm shift took place when cloned personal computers by several manufacturers resulted in IBM to lose its leadership, thus paving way for software to become the new light of industry – Microsoft and not IBM became the most important computer company.

Open source software today, acts as another paradigm shift. It is indeed threatening to change the rules of the computer industry.\(^3\) Our premise in the paper is that this may not necessarily be true in the same vein as IBM or Microsoft. Contemporary literature has come to believe that open source and closed source act as substitutes, or better still, rivals in the software industry today, and one’s success is another’s failure. We build a case for their coexistence in today’s society. This is even truer if we place in perspective the idea that new forms of collective actions emerge in the social climate of opposition, ambivalence and conflict.\(^4\)


\(^2\)Kuhn noted that scientific breakthrough “is seldom or never just an increment to what is already known. Its assimilation requires the reconstruction of prior theory and the reevaluation of prior fact, an intrinsically revolutionary process that is seldom completed by a single man and never overnight.” See id. at 7.

\(^3\)But see Anna Maria Szczepanska, Magnus Bergquist and Jan Ljungberg, High Noon at OS Corral: Duets and Shoot-outs in Open Source Discourse, in PERSPECTIVES ON FREE AND OPEN SOURCE SOFTWARE 431-46 (Joseph Feller et al., eds., 2005) (OSS movement can be related to the societal changes that began in 1960s: rise of network society supported by a plethora of communication and information technologies and the rise of new forms of collective action, which have also been referred to as new social movements). Similar undertones have been raised in several other works. See generally ALAIN TOURAIINE, THE VOICE AND THE EYE: AN ANALYSIS OF SOCIAL MOVEMENTS (1981); ALBERTO MELUCCI, CHALLENGING CODES: COLECTIVE ACTION IN THE INFORMATION AGE (1996); HAKAN THÖRN, MODERNITET, SOCIOLIOCHSOCIALARORELSER 62(1997). (Monograph from the Department of Sociology, Goteborg University). We have sympathies for arguments raised in above works, and we agree that these movements have in turn are a result of inter-relations of political, economic, cultural and institutional powers cross navigated from a modern to postindustrial society. However, that would hardly change the subject matter to possess an epochal shift from one form of action to another. Infact, it reinforces the power and structural potential of OSS even more.

\(^4\)See Melucciat 42-51.
Both IBM and Microsoft had altered the power game in information technology (IT) and the culture of IT industry, thus propelling and fuelling resistance to such polar digital world, not to mention attempts at bridging the digital divide. One result of the collective resistance was open source software project. In this backdrop, any argument favoring coexisting of open source and closed source (when former’s emergence is a result of resistance to closed source in the first place) may be welcomed with suspicion, but that’s precisely what we would like a reception for.

In order to understand why open source and closed source business structure of producing software can mutually exist, it is important to understand what open source and closed source structure signify. The term “source” in open source refers to source code. Programmers use a number of different programming languages to write source code, all of which are readable and understood by humans. In other words source code is the language that programmers use to speak to computers as well as command them to execute desired functions. A fundamental tenet of open source structure of producing software is that the user or licensee of the software must get both, the access to source code and more importantly the right to make changes to the source code and to customize programs or add features, as the user deems appropriate. On the other hand, most license agreements for closed source structure of producing software prevent the licensee or user from having access to the source code. One reason why closed source structure producing of software restricts access to source code is to protect the innovative and labour intensive parts of the program. The open source software movement, therefore, is often described as a fundamentally new way to develop software that poses a serious challenge to commercial software business.

---

5 For an illustrative backdrop of how contemporary movements in USA gave rise to the intellectual climate which paved way for ‘culture of IT’, see Manuel Castells, The Rise of the Network Society (2d ed. 2000).
6 See Jean-Jacques Gauguier & Rémi Douine (2005), Local Software and Local Content Production Challenge in Developing Countries. What can be learned from Open Source and Creative Commons Paradigms, 58 COMM. & STRATEGIES1 (2005).
10 See Nadan at 355.
11 See generally Chris DiBona et al., Open Sources: Voices from the Open Source Revolution (1999) (discussing the novel vision of the software industry that has been created through open source and explains how the movement works, why it succeeds and where it is going); see also Eric S. Raymond, The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary (2001).
12 See Paul Vixie, Software Engineering, in Open Sources: Voices from the Open Source Revolution 47-50 (Chris DiBona et al., eds., 1999).
Most precisely, open source software can be defined as software which provides the following minimal rights to its users: (a) right to make copies of the program and distribute those copies, (b) right to have access to software’s source code, and (c) right to make improvements in the program.\textsuperscript{13}

Although now new,\textsuperscript{14} this phenomenon of producing software based on unconstrained access to source code and the rapid growth of the open source structure of producing software was really fuelled not so long ago by Linux operating system\textsuperscript{15,16} and Apache web server\textsuperscript{17} have raised a series of important questions.

\textsuperscript{13}See Bruce Perens, The Open Source Definition, in OPEN SOURCES: VOICES FROM THE OPEN SOURCE REVOLUTION 79-86 (Chris DiBona et al., eds., 1999).
\textsuperscript{14}We do not intend to dwell in detail on historical path taken by open source movement and its development. However, for quick glimpse, following may suffice:
The first era, spanning from early 60s to early 80s focus on early developments in computer systems at Berkley and MIT and at research centers of Bell Labs, Xerox Palo Alto Research Center, etc., and different organizations commonly shared basic operating code of computer programs – source code. This era saw the growth of Unix and C language, while further innovations were made as the software was shared with others. Usenet was a computer programming network developed in 1979 to link together all users of Unix and this gave a huge impetus on development of software on the lines of OSS. This was all informal until early 1980s when AT&T began enforcing its (purported) intellectual property rights related to Unix.
The second era went on from early 80s until early 90s was categorized as the time when informal software development began formalizing its structure particularly after threats of litigation. Richard Stallman a MIT Artificial Laboratory set up the Free Software Foundation (FSF) in 1983. One major milestone in this era was the introduction of a formal licensing procedure by FSF that aimed to preclude assertion of patent rights for cooperatively developed software. GNU was a product of FSF, and had an interesting recursive acronym: “GNU’s not Unix.” The era also witnessed the General Public License (GPL, also known as copylefting) where user agrees to impose no restrictions on others. And every code had to be licensed on same terms (this is different from shareware where underlying codes are not made freely available). GNU project also developed a number of important of organizational features, particularly a model where contributions from many users were accepted but official version of the program was managed by smaller subset of individuals or in some cases, by individual leader.
The third era running from early 90s until this day is marked by widespread expansion of internet access and therefore a humongous increase in volume of contributions towards development of OSS. Linux is a result of this activity. This era also marked alternate licensing arrangements surfacing when only GPL was used earlier. In particular, Debian, an organization set up to disseminate Linux set up the “Debian Free Software Guidelines” in 1995, this allowing greater flexibility. These provisions were adopted in 1997 by a number of individuals involved in cooperative software development and were subsequently dubbed as “open source definition.” Past years have seen an unprecedented growth of OSS alongside increasing challenges.
This is built on Josh Lerner & Jean Tirole, Economic Perspectives on Open Source, in PERSPECTIVES ON FREE AND OPEN SOURCE SOFTWARE 50-52 (Joseph Feller et al., eds., 2005).
\textsuperscript{15}In 1991, about ten people were using and modifying Torvald’s original 10,000 lines of code; today there are an estimated seven million people using Linux and the code has grown to approximately 1.5 million lines.
Furthermore this open source operating system covers nearly 23 percent of the server operating system market. See Patrick K. Bobko, Linux and General Public Licenses: Can Copyright Keep ‘Open Source’ Software free? 28 AIPLA Q.J. 81, 82-85 (2000).
\textsuperscript{16}Linux, in particular, has an interesting history that augurs well with the spirit of open source project. Linux was initially developed by Linus Torvalds in 1991. It has been revised several times since then. This revision is done by a group of volunteers who communicate through the Linux kernel mailing list on the Internet. Torvalds
This article endeavours to clarify how open source structure and closed source structure of producing software can continue to mutually exist? Secondly, this paper also throws light on the much debated and discussed question as to why thousands of top-quality programmers contribute freely to development of open source, which is a public good.

This paper is organized and structured into three sections. In the first section this article articulates the ideological and ethical stance of open source advocates and closed source proponents. In the second part this paper discusses the theoretical insights concerning the motivational problems in the open source structure of producing software at the inventive stage. In the final section this paper briefly describes as to why open source structure will coexist with closed source structure of producing software?

I. OPEN SOURCE ADVOCATES V/S CLOSED SOURCE PROPOONENTS

The technical advantages and disadvantages of open source vis-à-vis closed source don’t merit our attention in the paper. However, it is imperative that a socio-economic comparative perspective be laid bare for a better understanding of the debate. Perceptions matter and in this section, we intend to categorize rather semi-porous groups having differential perceptions about the underpinning philosophy entailed in the open source v. closed source debate. There is a wide range of views regarding software ownership, even among the information technology professionals. The hacking community and even within the hacking community there are clear differences of opinion concerning Intellectual Property and

has acted as the main kernel developer and exercised some control over the development. Commercial companies have added value packaging the code and distributing it with documentation. Linux is a Unix-like operating system. Linux was initially developed for Intel 80386 microprocessor. Over the years, developers have made Linux available on other architectures. Most of the platform-dependent code was moved into platform-specific modules that support a common interface. Linux is a kernel; it does not include all the applications such as file system utilities, graphical desktops (including windowing systems), system administrator commands, text editors, compilers, and the like. However most of these programs are freely available under the GNU (GNU is a recurring acronym for GNU’s not Unix, when Unix, which was developed as open source) General Public License and can be easily installed in a file system supported by Linux. See generally DANIEL PIERRE BOVET & MARCO CESATI UNDERSTANDING THE LINUX KERNEL (1st ed. 2000). See also (for more technical yet lucid details)W. RICHARD STEVEN, TCP/IP ILLUSTRATED: THE PROTOCOLS, VOLUME 1 (2000); DOUGLAS COMER, INTERNETWORKING WITH TCP/IP: PRINCIPLES, PROTOCOLS, AND ARCHITECTURE, (1998).

As per the most recent survey by Netcraft more than 60 percent of total servers employ Apache, rather than commercial alternatives from Microsoft and other firms; see http://news.netcraft.com (last visited on June 25, 2007).

software. McGowan et al.¹⁹ have identified four distinct subgroups of people who by choice associate themselves with information technology and describe their positions on intellectual property. The first subgroup, which does not identify with the hacking community, is labelled as closed source proponents. The other subgroups, which have ties to hacking community, are labelled as open source advocates, free software ideologues and software anarchists.

**Socialism**

<table>
<thead>
<tr>
<th>Software Anarchists (SA)</th>
<th>Free Software Ideologues (FSI)</th>
<th>Open Source Advocates (OSA)</th>
<th>Closed Source Proponents (CSP)</th>
</tr>
</thead>
</table>

Source: McGowan et al. 2004

**Software Anarchist (SA):** If a person sees himself as separated from mainstream society and the business-oriented view of software, then he is more likely to be a Software Anarchist.²⁰ Jordan and Taylor²¹ go on to explain that Software Anarchists have an addictive relationship with technology. They are consumed by technology and their addiction leads them to believe that all technology should be available for them to dissect and understand which leads hacking and cracking both unethical and illegal in the eyes of mainstream society.²²

**Free Software Ideologues (FSI):** The basic ideology of FSI is that, neither individuals nor corporations should develop computer software using proprietary code. The positive aspect of the free software philosophy advocated by FSI makes it clear that “free” means software that is free to be changed, to be improved and better understood through available open source code.²³ This idea was proposed by Richard Stallman, who founded the Free Software Foundation in 1985, which intends to preserve free access for all the software developed by software hackers.²⁴ When FSIs are talking about “free” software they are positioning the writing of software code as free speech. Coleman²⁵ writes that justification for this belief stems from the argument that a free software system develops intellectual goods that are

---

²⁰Id. at 410.
²²Id. at 766.
²³See supra note 20 at 412.
²⁴Richard Stallman was a brilliant programmer and was housed at MIT’s Artificial Intelligence Laboratory from 1960-70. In 1980s this group along with Richard Stallman received a major jolt when MIT licensed some of the code created by its hacker employees to a commercial firm. This firm promptly restricted access to the “source code” of that software and so prevented the MIT hackers from continuing to use it as a platform for further learning and development; see STEVEN LEVY, HACKERS: HEROES OF COMPUTER REVOLUTION (1984); see also GLYN MOODY, REBEL CODE (2001); see also Georg Krogh & Eric von Hippel, Special Issue on Open Source Software Development, 32 RES. POL’Y 1149, 1150 (2003).
superior and benefit society as a whole. The basic license developed by Stallman to implement this idea or belief was the General Public License (GPL), which is sometimes referred to as “copyleft”.26 According to General Public License those possessing a copy of free software have a right to use it at no cost, the right to study its source code, to modify it and to distribute the modified or unmodified versions to others at no cost.

**Open Source Advocates (OSA):** Open source advocates promote the open source philosophy, according to which, all software should be developed in an environment where programmers can read, redistribute and modify the source code. They also argue that the greatest good is served by having a system that allows anyone to modify software. By permitting anyone to improve a software product, everyone will benefit from the improvements. The ideological groundings of free software movement and open source movement are one and the same. The differences are in their approach to promoting their ideas and their willingness to compromise these ideas in order to gain more widespread acceptance. Schimdt27 has described the differences between the OSA and FSI subgroups. According to him, the most significant difference is that the OSAs are willing to let go of free speech arguments and some of the positions held by FSIs in an effort to find common ground with the corporate world.28 Eric Raymond who is the vocal champions of open source and the president of the OSI board believes strongly that there are both quality and economic reasons for developing software this way. Many scholars29 have conducted research to test the arguments put forth by OSAs in an attempt to explain why a wide variety of useful and reliable software models can emerge from the open source model. Faldetta30 suggests that a person would contribute to development of software in the open source structure to satisfy his own needs as well as the needs of others. He further argues that the software produced in the open source is better than the software produced in closed source. Furthermore open source advocates also believe that the current IP system has become untenable, and that it impedes the rapid development of quality software.

**Closed Source Proponents (CSP):** Closed source proponents have formulated the following arguments against the open source structure of producing software. Firstly, there will be little incentive to create software if one cannot profit from it; and secondly, it is not fair to the creator to put effort into creating software and not profit from it. Furthermore the followers of closed source structure believe that persons who create software acquire rights of ownership

---

28*Id.* at 680.
in the software by expanding labour in its development. The ideology of the closed source proponents revolves around the fact that the creator who expands effort reaps the rewards of such ideas and users who build on such ideas must obtain permission from the creator and make appropriate fair payment. Richard Epstein who is a strong supporter of property rights argued that the open source system of producing software might work well in the early days when the programmers remains fixed. But when a given programmer wants to quit, that programmer does not receive in cash or kind his share of the gain in value during this period of coding. If this is the case then the run-up in value during his period of coding will be gobbled up by his successor, which turns out to be a recipe for immense resentment. This argument of Epstein can be assessed and reviewed only by studying the motivational concerns in the open source structure of producing software.

II. WHAT MOTIVATES THE OPEN SOURCE PROGRAMMER TO CONTRIBUTE SOFTWARE FOR FREE

To an economist, it is surprising to witness the behavior of so many individual programmers who work on open source. It is unclear as to why should these programmers freely contribute to the provision of a public good. Following comments from two leaders of open source movements have noted the existing system, albeit not purely from an economist’s angle:

The idea that the proprietary software social system – the system that says you are not allowed to share or change software – is unsocial, that it is unethical, that it is simply wrong may come as a surprise to some people. But what else can we say about a system based on dividing the public and keeping users helpless?

The “utility function” Linux hackers are maximizing is not classically economic, but is the intangible of their own ego satisfaction and reputation among other hackers. [Parenthesis comment deleted.] Voluntary cultures that work this way are actually not uncommon; one other in which in have long participated in science fiction fandom, which unlike hackerdom explicitly recognizes “egoboo” (the enhancement of one’s reputation among other fans).

We discuss what motivation may lie for programmers to invest their time and efforts in contributing to a public good, like open source. Social psychologists and Sociologists have conceptualised the motivational processes of persons under two categories namely, Intrinsic and Extrinsic. Koch in his influential work presented a logical description of work

31 See DEBORAH JONHSON, COMPUTER ETHICS (1994)
32 Id. at 33.
34 The argument of altruism is not convincing to say the least. It is least emphasized and has not been seen in any other industry just so that a case can be built for IT industry.
35 Richard M. Stallman, The GNU operating system and the free software movement, in Chris Dibona et al. supra note 11 at 31-8.
36 Raymond, supra note 11 at 564-565.
behaviour and accordingly described two conditions of work motivation. The first condition of work motivation encompasses the expression of persons with the feeling that a sense of one’s behaviour is regulated by an external source, the external source being required to recognise and reinforce the behaviour.\(^{38}\) The second condition of work motivation encompasses the expressions of persons who feel they have freely invested themselves in their work.\(^{39}\) Drawing inferences from Koch’s discourse, De Charms\(^{40}\) proposed that whenever an individual perceives the locus of causality of task behaviour to be external to him, he would consider himself to be extrinsically motivated. When the individual experiences to be the locus of causality of one’s own behaviour he would consider himself to be intrinsically motivated. Deci in his seminal works has emphasized that,

\[\text{Intrinsic work motivation is a cognitive state reflecting the extent to which the worker attributes the force of his or her task behaviours to outcomes derived from the task per se and Extrinsic work motivation as a cognitive state reflecting the extent to which the worker attributes the force of his or her task behaviours to having and expecting to receive or experience some extrinsic outcome.}^{41}\]

The above distinctions between motivations which are rooted in the psychology of the individual (internal factors) and rewards (external factors) will be used for analysing the factors that lead programmers to participate in open source development. Many empirical studies\(^{42}\) have confirmed that both intrinsic and extrinsic motivational factors play a role in open source structure of producing software.

**Intrinsic Motivation**: Applied to open source environment, the intrinsically motivated programmers are defined to include programmers who are motivated by the feeling of competence, satisfaction and fulfilment that arises from writing programs.\(^{43}\) Another variant of intrinsic motivation is altruism\(^{44}\), which is defined as doing something for another at some cost to oneself; for e.g. open source programmers provide software for others at their own cost of time, energy and opportunity cost. Perkins\(^{45}\) argues that the production of open source

---

\(^{38}\) Id. at 17.

\(^{39}\) Id. at 17.

\(^{40}\) See RICHARD DE CHARMS, PERSONAL CAUSATION (1968).


\(^{44}\) See JAMES R. OZINGA, ALTRUISM (1999).

\(^{45}\) See G. Perkins, *Culture Clash and the road of World Dominance*, 16 IEEE SOFTWARE 80 (1999).
software is a form of intellectual gratification with an intrinsic utility similar to that of a scientific discovery. In addition to the above papers which have addressed the motivation for the individual to take part in the development of open source software, Bitzer et al. have adapted a dynamic private-provision of a public good model to reflect key aspects of the open source phenomenon, such as play value, user-programmers and gift culture benefits contribute to the economic analysis of open source development.\(^{46}\)

**Extrinsic Motivation:** Open source structure of producing software may also be motivated by external factors. Open source programmers by participating in open source projects can expand their personal skills, capabilities and knowledge and this is deemed as special forms of human capital by economists.\(^{47}\) According to Gary S. Becker education, training and learning leads to increase in one’s human capital, which shows the way for better job opportunities, higher salaries and more fulfilling jobs.\(^{48}\) On the other hand, the explanation suggested by Lerner and Tirole\(^{49}\) is that, programmers are writing codes to contribute to open source projects in order to signal their ability to future employees, venture capitalists or to peer and thereby boost their human capital or get ego gratification. Hann et al. examining the longitudinal data set of participant programmers identified that their results are consistent with the notion that a high rank within the Apache Software Foundation is a credible signal of the productive capacity of a programmer.\(^{50}\) The essential facet, which follows from the above study, is that working on open source projects provides the prestige, visibility and acts as a medium which distinguishes the open source programmers from other programmers and thereby gives the open source programmers better chance to be noticed by software firms.

A familiar and simple way of application of utilitarian perspectives to assess motivation of programmers, we can include factors similar to those assessed by Lerner and Tirole.\(^{51}\) The net benefit in the long run (two time periods)\(^{52}\),

\[
\prod_{t} \Pi_{n, c, t} = \prod_{t=0}^{1} \pi_{c, t} + \sum_{t=0}^{1} \pi'_{c, t}
\]

where, \(t\) refers to time period and is defined as \(t \in (0,1]\) for current and \(t \in (1,2]\) for delayed time period.\(^{53}\) Each of the payoffs \(\pi_c\) and \(\pi'_c\) can be symmetrically represented as difference of benefits and costs denoted by \(\alpha\) and \(\beta\) respectively. Hence,

\[
\pi_c = \pi_{c, t} - \pi_{c, t}'
\]


\(^{48}\)Id. at 11.

\(^{49}\)See Lerner & Tirole, *supra* note 29, at 215.


\(^{51}\)See Lerner & Tirole, *supra* note 14, at 56-59.

\(^{52}\)We express our long run equilibrium to consist of two time periods – current and delayed.

\(^{53}\)can be safely assumed to have normal distribution in the specified interval but this assumption does not help us especially.
\[
\pi^*_t = \phi^*_t - c^*_t
\]

Table 1 expresses some factors for the respective costs and benefits. (Note that costs remain the same in both time periods).

<table>
<thead>
<tr>
<th>Current time ( t \in [0,1] )</th>
<th>Delayed time ( t \in (1,2] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi_t ) self-improvisation in programming abilities</td>
<td>( \phi_t ) fun opportunity cost of time</td>
</tr>
<tr>
<td>( c_t ) opportunity cost of time</td>
<td>( c_t ) monetary compensation (if independent)</td>
</tr>
<tr>
<td>( \phi_t ) monetary compensation (if independent)</td>
<td>( \phi_t ) career incentive</td>
</tr>
<tr>
<td>( c_t ) opportunity cost of not focusing on primary mission (if employed)</td>
<td>( c_t ) ego gratification opportunity cost of not focusing on primary mission (if employed)</td>
</tr>
</tbody>
</table>

Table 1: Relative merits and demerits of open source in two time period model

Note that the factors of costs remain the same except may be their quantum. With discounting future benefits and costs, we can only safely assume that the costs actually go down as a programmer moves from time period one to time period two. But changes in benefits and their analysis seem important here.

Both incremental benefits in form of career concern incentive and ego gratification are similar from economic perspective and can be grouped together under signalling incentive, if looked under the lens of economics. This is precisely what we build our model’s basis on. Career concern incentives refer to future job offers, shares in commercial open source-based companies\(^54\) or even future access to venture capital markets.\(^55\) The ego gratification stems from recognition amongst peer group. Economic theory\(^56\) suggests that the collective incentive, called signalling incentive is stronger, the more visible the performance is, the

---

\(^54\) Linus Torvalds and others have been awarded shares in Linux-based companies that went public. Although these rewards were unexpected in their entirety and there is little reason to suspect that they tinkered with the motivation of programmers, however, there is no reason to believe that if such awards are institutionalised in future, their expectations will not increase and that it will not impact the motivation of the programmers or open source leaders. The private benefits of leadership seem to grow in importance as sector or programmer matures (time two). See, Lerner & Tirole, supra note 14, at 61-2. See also, Lerner & Tirole, supra note 29.

\(^55\) Venture Capital market has been the fuel supplier, and in many ways, the financial backbone of many IT products, both software and software. In fact, it would not be hard to imagine a very insulated world as that which existed before 1990s, had it not been for venture capitalists to ensure that the ideas of programmers saw the light of market and could cross navigate across countries. See, for e.g., Richard Coopery, Venture Capital and Computer Industry: Financing Growth Companies in UK, 24 BUS. & ECON. HIST. 266 (1995) (particularly for UK). See also, Bob Zider, How Venture Capital Works, HARV. BUS. REV. Nov.-Dec., 1998, at 131-40.

\(^56\) See, for e.g., B. Holmström, Managerial incentive problems: A dynamic perspective, 66 REV. OF ECON. STUD. 182 (1999).
higher the impact of effort on performance is and finally, the more informative the performance about talent is.\textsuperscript{57}

Signalling incentives give rise to ‘strategic complementarities’ and programmers look out for working on programmes that they expect to attract large number of ‘audiences.’ Open source in this respect is not very different from academic research, which is well known to exhibit fads.\textsuperscript{58} There are evidences that there are areas which are completely research-malnourished for years while those over-fed are becoming obese due to their attraction by large number of researchers. A fad can create strong signalling incentive because researchers working in popular areas have sufficient motivation to produce a high quality work because they expect it to be seen by large group of audience.\textsuperscript{59} One can however think that this does not explain leader’s motivation in revealing the source to public and forego substantial career and monetary rewards of an important open source project. One possibility is that leader’s own corporation may act as natural resistance to his/her technology.\textsuperscript{60} This is because of an absolute need for procuring a license for the creative work form the employer if the work was done while working in-house and a company would not want to lose a capable employee and therefore deny her/his request.\textsuperscript{61} Also, there are theories to suggest that innovation is a result of fighting against a dominant player in the industry.\textsuperscript{62}

It may be important to appreciate the differential structure of incentives in open source and closed source. Clearly for short term $t \in (0,1]$, closed source (due to its proprietary nature of code) offers monetary compensation in form of income, which acts as a very strong incentive for employed programmers. This is the classic justification in favour of strengthening of intellectual property for increasing innovation rate. By contrast, an open source may lower costs too, on account of the reduced cost since open source is free and therefore has higher availability, thereby making it more familiar to programmer and making their cost of working on open source less costly. Secondly, there are customization and bug-fixing as purely private benefits for the programmer. The delayed benefits, signified as signalling incentives appear to be stronger in open source. This may be because of (a) better performance visibility in open source: if the program is closed, outsiders can hardly measure the programmer’s performance, and can make more informed decision about ability of the programmer,\textsuperscript{63} (b) self-ruler: programmer in open source setting is his/her own boss and it is upon his/her full initiative that the project kicks off, as opposed to a traditional hierarchical commercial firm where

\textsuperscript{57}See, Lerner & Tirole, supra note 14 at 56-58.

\textsuperscript{58}See, ROBERT K. MERTON, THE SOCIOLOGY OF SCIENCE: THEORETICAL AND EMPIRICAL INVESTIGATIONS (1973) (for an excellent discussion on historical examples on simultaneous discoveries).

\textsuperscript{59}See, Partha Dasgupta and Paul A. David, Towards a new economics of science, 23 RES. POL’Y 487 (suggesting alternate solution, albeit with no special confidence that the argument mentioned here is incorrect or even remotely unreasonable).

\textsuperscript{60}See, Lerner & Tirole, supra note 14 at 64-65.

\textsuperscript{61}Id. at 65.

\textsuperscript{62}See, Thörn, supra note 3. See also Melucci, supra note 3.

\textsuperscript{63}Lerner & Tirole, supra note 14 at 60.
supervisors’ interference and advises dampen the enthusiasm of a young and brilliant programmer, and finally, (c) higher fluidity: the labor market is more fluid in open source environment.

These arguments build a case to highlight as to who is more likely to contribute and what tasks are best suited to open source projects. There are two categories of people which emerge with propensities towards open source as against closed source. First ones are those who are more sophisticated since they derive direct private benefits when they customize or fix a bug in an open source. The second group comprises of those individuals who have strong signalling incentives and who are looking forward to use open source as part of entry either into the group(s) of academic institutions, particular peer groups, prospective employers, venture capital community and so forth.

Before moving on to the next section, we need to give a brief word of caveat in viewing open source as just another incremental innovative activity. Technological innovation in the past has some interesting parallels with open source evolution where many other scientific discoveries were a result of sophisticated users. This has also been the case in early days of computer industry when innovations by particular users for individual problems became more general solutions for wide class of users. In a different set of literature, authors have also argued that adoption of scientific institutions within for-profit organizations. In addition, studies have particularly noticed the extent to which encouraging employees in pursuing both basic and applied research have led to substantial challenges in designing incentive schemes due to the different output of each activity and means through which performance is measured. Open source on the other hand is quite distinct from earlier activities, especially in the extent to which contributor’s work is recognized and rewarded. Moreover, since this is an ongoing phenomenon, it stands as a potential threat to an existing mammoth structure of software programming currently under the purview of intellectual property rights. Open

64 And it is also in the case of self and full initiative that the performance is more precisely measured. See, Jaime Ortega, J. Power in the firm and managerial career concerns (Universidad Carlos III de Madrid Working Paper, 2000) (for empowerment and career concerns). See also, Bruno Cassiman, The organization of research corporations and researcher ability (University Pompeu Fabra Working Paper, 1998) (for another discussion on how it is difficult to sustain a reputation for respecting autonomy of researchers within firms).
65 Lerner & Tirole, supra note 14 at 60.
66 See for a short discussion id. at 62-72.
source opens a plethora of legal, social and economic questions that have people on both sides of the debate. There is hardly any consensus on what should the optimal solution be, but in the same vein, there is hardly any doubt on the merit of considering open source distinctly from any other incremental innovation that may have taken place in history.

III. OPEN SOURCE AND CLOSED SOURCE BUSINESS STRUCTURE CAN MUTUALLY EXIST!

Open source and closed source structure will co-exist, because if open source system acts as a substitute for the closed source commercial software then the contribution of talented programmers to open source software development will end up being lower than in case where open source structure and closed source structure mutually exist. The reasoning is quite simple, open source business structure has evolved and has contributed significantly to the growth of highly skilled programmers by discerning them from other low level programmers. Therefore open source business structure will continue to exist flanking closed source business structure until the highly skilled programmers feel a need for demarcation from less skilled programmers. There will always be low level programmers for whom it is not advisable to choose an open source career and thus voluntarily choose closed source structure. If no such need for demarcation is felt, then every programmer will choose to be a closed source programmer anyway.

It is our assertion that there will always be a need for such demarcation, which is perceptible from the projects they work or programs they contribute. To make this demarcation very obvious the open source programmers do not perform the less prestigious, but just as important, development tasks such as documentation, designing easy-to-use interfaces, providing technical support, and ensuring backwards compatibility.

To examine as to how open source structure of producing software will mutually exist with closed source system of producing software, it becomes important to analyze as to what is the driving force for highly skilled programmers to participate in the open source development. It is our argument that highly skilled programmers firstly participate in open source development to expand their skills and secondly to differentiate the so acquired skills from less skilled programmers from producing complex programs.

---

72 Less attractive projects either fade with neglect or be maintained by less skilled developers, see Carolyn A. Kenwood, A Business Case Study of Open Source Software (MITRE Working Papers available at http://www.mitre.org/work/tech_papers/tech_papers_01/kenwood_software/index.html (last visited 11th January 2008)).
Participation in Open Source development act as a signal of ability: Let us assume that programmers can expand their skills by participating in open source projects and thus produce complex products. Plausibly the programming levels so acquired may not have any bearing at all on the future job requirements of the programmer, which he is expected to take, but nevertheless, the programmer needs to acquire these programming levels to prove his abilities to the potential employers. This paper intends to capture this effect in what follows. Note that signaling as one of the most pivotal driving factor is also witnessed in practice. Evidence that contributions to open source software developments are being driven by signaling and reputation concerns can be found in the analysis of contributors to a long-standing archive of Linux postings maintained at the University of North Carolina. Existence of signalling and recognition effects has indeed several other evidences. It is clear that giving credits to authors is essential in the open source movement. Reputational benefits that accrue from successful contributions to open source have real effects on developers. Primary benefits are good reputation among peers, attention and cooperation from others, higher status, and the likeness. Founders of Sun, Netscape, Red Hat and many more have signalled their talent in the market. Apache project provides good illustration of the same too, which recognized all its contributors on its website with special highlighting made to most committed contributors. And many of these programmers have been hired into big commercial companies like IBM, Collab.Net and many more.

The reasoning could be that the cost of acquiring good or quality programming levels is much higher for programmers of low ability than it is for programmers with high ability; the effect is that programmers with low ability cannot profitably imitate programmers of high ability. Thus acquiring good or quality programming levels signal high ability of the programmer, even if the programming community does not foster that ability. Potential employers know that the programmers acquiring good or quality programming levels by participating in the open source development possess high ability because only for such programmers it is worthwhile to obtain these levels. Here is a simple model presented to study and understand this logic.

A programmer’s ability, which is either \( H \) or \( L \), where we assume that \( L < H \), is known only to the programmer but not to the potential employer. The programmer chooses the amount \( p \) of programming level to obtain and then the employer, observing \( p \), offer wage \( w \). Let us assume that acquiring a programming level \( p \) is less expensive to obtain for a programmer of high

\[74\text{From various studies on open source programming, we learn that programmers participate in open source projects to expand their programming skills and develop complex products. Hence, it is reasonable to assume the above; see Fitzgerald & Feller; von Hippel; Lerner & Tirole supra note 29. See also, Hertel, supra note 42; Hars & Ou, supra note 43; Bitzer, supra note 46.}

\[75\text{See, B.J. Dempsey, D. Weiss, P. Jones and J. Greenberg,} \textit{A quantitative profile of a community of open source Linux developers} \text{(School of Information and Library Science, University of North Carolina at Chapel Hill Working Paper, 1999).}

\[76\text{This principle is included as part of nine key requirements in the ‘Open Source Definition.’ See also, Raymond, supra note 11.}

\[77\text{Id.}\]
ability than it is for a programmer of low ability. Specifically, assume that the cost to a programmer of ability $H$ of obtaining $p$ units of programming levels is $p/H$ and the payoff for the programmer who obtains $p$ units programming levels and takes the job paying $w$ is $w-p/H$. The payoff for the employer who pays the programmer of ability $H$ the wage $w$ is $H-w$. The extensive game that corresponds to this model is given in below Figure II.

I aver that the game has a weak sequential equilibrium in which a high-ability programmer chooses a positive amount of programming level, which will help him in producing complex programs. Deeming the following assessment in which $p^*$ is a positive number.

**Programmers Strategy:** Type $H$ programmer chooses $p = p^*(open$ $source)$ and type $L$ programmer chooses $p = 0$ ($closed$ $source$); after observing the employer wage offers, both types choose the highest offer.

**Employers Belief:** Employer believes that a programmer is type $H$ if he chooses $p^*(open$ $source)$ and type $L$ if he chooses otherwise ($closed$ $source$).

---

**Figure I:** This diagram shows an outline of an extensive game that models programmers who signals his ability to a given employer by acquiring one of the possible programming levels $p$ and the employer making the wage offer $w$. The programmer payoffs are listed first and the employer payoff second. Following the assessment in which type $H$ programmers chooses $p = p^*$ and type $L$ programmer chooses $p = 0$; the payoffs of the programmers are given accordingly in bold. (See Spence, infra note 87.)

Here the employer is moving and believes that a programmer is type $H$ if he chooses $p^*(open$ $source)$ and type $L$ if he chooses otherwise ($closed$ $source$). It is pertinent observe that still the information is incomplete in the connected nodes.

Case I: The low ability programmer mimics high ability programmer by choosing $p = p^*$

Case II: The high ability programmer mimics low ability programmer by choosing $p = 0$
Employers Strategies: Employer offers wage $w_H$ to a programmer who chooses $p^*$ (open source) and the wage $w_L$ to a programmer who chooses any other value of $p$ (closed source).

The employer’s beliefs are consistent with the programmer’s strategy (i.e. no programmer chooses a programming level different from $p^*$ and $0$). Parameters under which the players strategies are sequentially rational, let’s consider the programmers initial action.

Type H: If the employer offers wage $w_H$ to a programmer who chooses $p^*$ and wage $w_L$ to a programmer who chooses any other value of $p$, then the type $H$ programmer obtains the payoff $w_H p^*/H$ only if the programmer follows his strategy and chooses programming level $p^*$. A type $H$ programmer obtains the payoff $w_L p^*/H$ if he chooses any other programming level $p$. The programming level $0$ achieves the highest payoff of $L$ for a deviant so for the equilibrium we need $w_H p^*/H \geq w_L$ or $p^* \leq H (w_H - w_L)$.

Type L: A type $L$ programmer obtains the payoff $w_L$ if he chooses his strategy and chooses his programming level $0$. If he chooses any programming level other than $p^*$ he obtains the same wage, and pays a cost, so such a deviation is not profitable. If he chooses $p^*$, then the employers believe his ability to be $H$ and he obtains the payoff $w_H p^*/L$. Thus for equilibrium we need $w_L \geq w_H p^*/H$ or $H (w_H - w_L) \leq p^*$.

Employer: Employer’s payoff is 0, given its belief and its strategy. If it increases the wage it offers in response to any value of $p$, its expected profit is negative, given its belief and if it lowers the wage its expected profit remain 0 (its offer not accepted).

In summary the assessment is a weak sequential equilibrium if and only if $L (w_H - w_L) \leq p^* \leq H (w_H - w_L)$. We have assumed that $L < H$, so left hand side of the above expression is less than the right hand side and hence $p^*$ satisfying the expression exists. In other words for any value of $H$ and $L$, the game has a separating equilibrium in which high ability programmers obtain some programming levels by participating in the open source movement, whereas low-ability programmers do not. A high ability programmer obtains the said programming level to avoid being labelled low in ability by potential employers. In this game the strategic interest of the high ability programmers is to acquire programming levels produce complex programs. In other words high $p$ also means higher ability to produce complex programs.

What follows from the above assessment is that high ability programmers participate in the open source development to expand their programming levels to produce complex programs and in doing so provide stronger visibility for their signals. This assessment also stands well supported by Lerner and Tirole’s supposition that programmers are willing to contribute to open source projects in order to signal their ability to future employers. They also identify quite a few reasons for strong signals from open source programmers; Firstly, performance measurement is better under open source. This is because only the functionality and usage of

78See Lerner & Tirole supra note 29 at 212.
a closed source program can be observed by the outsiders, while under open source the contribution of each individual and the quality of his or her code can be directly reviewed. Secondly, an open source programmer takes full responsibility for his or her project or sub-projects, whereas in a traditional firm environment, an individual’s performance depends on that of others. Lastly, open source programmers are less likely to have firm-specific human capital. Leppämäki and Mustonen have also considered a model in which programmers signal their talent to software firms by choosing how many lines of code to contribute to an open source project. Furthermore Lee et al. have also considered a model in which programmers need to choose between joining closed source software firms or open source projects. If they join open source projects, they forgo current wages, but can signal their productivity to software firms and hence boost their future wages. From our analysis and previous studies we can conclude, that open source projects acts as a well-developed mechanism to develop quality skills by developing complex products and thereby transfer these signals to the surrounding environment.

Whether co-existence of open source and closed source programmers is an evolutionarily stable strategy?

It is important to assert the homogeneity of the programmers’ groups and that they come from same category (this may be a difficult assumption for many). There are reasons to believe that open source process is quite elitist. An analysis of 25 million lines of open source code constituting 3,149 projects suggests that even though more than three quarters of nearly 13,000 contributors made only one contribution and only one in 25 had more than five contributions, the top decile of contributors accounted for fully 72 percent of code contributed to open source projects and top two deciles for 81 percent. This could be more skewed if reported ‘bugs’ are also taken into account. For every code-contributing individual, there will

---

79 A close inspection reveals that open source structure is organized such that every significant contribution can be traced back to the original author. Moon and Sproull identify that, in Linux Kernel, which is one of the biggest open source projects, there exists a public change log file which lists the names of the programmers who have contributed to the official source and their specific inputs. The authors name and contribution recorded in the change log file is an honor and sign of expertise amongst the programmers. See Jae Yun Moon & Lee S. Sproull, Essence of Distributed Work: The Case of the Linux Kernel, http://www.firstmonday.org/issues/issue5_11/ accessed on September 15, 2007; see also Eric S. Raymond, Homesteading the Noosphere, http://www.firstmonday.org/issues/issue3_10 raymond/index.html accessed on September 26, 2007.

80 See Leppämäki & Mustonen, supra note 71 at 9.

81 See S. Lee, N. Moisa & M. Weiss, Open Source as a Signaling Device- An Economic Analysis (Working Paper Series Finance and Accounting, Johann Wolfgang Goethe-University, Frankfurt/Main, 2003).


be five who will merely report errors. Interestingly, this is largely how contributions in closed source are also scattered and the same skewness of output is observed among programmers employed in closed source. This builds a case for a rather homogeneous group from where these programmers emerge.

Let’s consider a simple hawk and dove game to elucidate the question discussed in preceding section. Two programmers of the same category compete for a job (job which is highly rewarding) whose value is \( v \gg 0 \) (in this case \( v \) measures the expected increase in rewards). Each programmer can be either aggressive open source programmer with probability \( \pi \) who participates in the production of open source development of software or passive closed source programmer with probability \( 1 - \pi \) who participates in the closed source development of software.

If both programmers are aggressive, they will compete until one is selected for the available job; the winner obtains the resource or job in the present case without sustaining any cost, whereas the loser suffers a loss of \( c \) (\( c \) measures the cost incurred). If both the programmers are aggressive or open source programmers, both of them are equally likely to win, so each one’s expected payoff is \( \frac{1}{2}v + \frac{1}{2}(-c) \) or \( \frac{1}{2}(v - c) \). If both the programmers are passive or closed source programmers then each programmer obtains the resource or the job in the instant case with probability \( \frac{1}{2} \) of \( v \) or \( \frac{1}{2}w \). Finally, if one programmer is aggressive or open source programmer while the other is passive or closed source programmer, then the open source programmer obtains the specified job whose value is \( v \) without any competition. The game is shown in the Figure II.

If \( v \gg c \) then the game has a unique Nash equilibrium (NE) \((\text{Aggressive, Aggressive})\), which is also strict and accordingly strategy \( \text{Aggressive} \) becomes the unique evolutionary stable action. In other words, the programmers choose to be aggressive or open source programmer.

If \( v = c \), then also the game has a unique Nash equilibrium \((\text{Aggressive, Aggressive})\). But in this case the equilibrium is not strict, against an opponent that chooses strategy \( A \), because a player obtains the same pay-off whether it chooses \( \text{Aggressive or Passive} \).

---

Nevertheless, in both cases, a population of passive programmers can be invaded by aggressive programmers: An aggressive programmer does better than a passive programmer when its opponent is passive and at least as well as a passive programmer when its opponent is aggressive.

However, the interesting case is when $c < v$. If the population of programmers is homogenous (no different ability or no way to signal it to others), only the symmetric equilibrium in mixed strategies, where all players play “aggressive” with probability $\pi = v/c$, is evolutionary stable. In this case there is co-existence of both types of programmers, provided that the closed source business structure is not thrown out of the market by open source companies. Assuming that the population of programmers is heterogeneous (asymmetric) let’s say one can differentiate between the high ability programmers to less skilled programmers, then high ability programmers will announce they will play aggressive. In this case the unique equilibrium will be one where all high ability programmers will play aggressive and less skilled programmers will play passive. Co-existence of firms is again guaranteed but the game has a pure strategy NE.

From hawk and dove game it can be explained that being an open source programmer is evolutionarily stable action in all cases where $v \geq c$. However in cases where $v < c$ and the population of programmers is homogenous then the game has no symmetric NE in pure strategies: neither (Aggressive, Aggressive) nor (Passive, Passive) is a Nash Equilibrium (NE) but there exists symmetric equilibrium in mixed strategies. In cases where $v < c$ and the population of programmers is heterogeneous the game has a pure strategy NE, where high ability programmers play (Aggressive, Aggressive).
IV. CONCLUSION

It is highly unlikely that the readers of this article are homeless. None of you must be sleeping on the streets, even though we may be in shared apartments, rented places, dorms, hotels or may be at friend’s place. We are all in a place called property, property of someone if not ours. And that someone has the right to exclude. Then what would one say about the roads leading us to that place or about the highway leading to that road? No one has any right to remove us from that road or highway because they are common property. And yes, both the private and common property coexist in the world, in fact, their coexistence is pivotal for the modern societal structure to exist.  

When Auguste Kerckhoffs wrote about open versus closed system 128 years ago, he wisely assumed that the enemy knew one’s cipher system, so that the security could reside only in the key. The debate today has developed to an inconclusive argument as to whether access to source code of a software product is of more help to the defence, because they can find and fix bugs more easily, or to attackers, because they can develop exploits with less effort. In a perfect world and for systems large and complex enough for statistical methods to apply, the attack and the defence are helped equally, and hence, whether systems are open or closed makes no difference in the long run to the system. Ross Anderson explores this and concludes that there are enough deviations from this general principle that is applicable only in an ideal world. It is in the ideal world that our model fits. Several points of departure are highlighted by Anderson in his paper, and many of these points appear to dwell on our basic assumptions, thereby making our model more robust. However, the level of synchronization and whether there is an equilibrium achieved in our and Anderson’s paper is a work for future.

The themes of open source projects that feature in various academic sources usually harp on quality and reliability of open source software, its rapid release schedule and its availability at little or no cost. In addition, it has been suggested that this initiative has the potential to become the dominant form of work for knowledge-workers in the information society. Hence it becomes imperative for social scientists to model the motivations at work and propose a sustainable solution for driving forces of this system in future.

87 This analogy is derived from the keynote address of Lawrence Lessig (the then Jack N. and Lillian R. Berkman Professor for Entrepreneurial Legal Studies at Harvard Law School), Free Software – a Model for Society, 1 June 2000, Tutzning, Germany.
88 Auguste Kerckhoffs, La Cryptographiemilitaire, 9 JOURNAL DES SCIENCES MILITAIRES5 (1883).
89 See, Ross Anderson, Open and Closed Systems Are Equivalent (That is, In an Ideal World), in PERSPECTIVES ON FREE AND OPEN SOURCE SOFTWARE 127-142 (Joseph Feller et al., eds., 2005).
90 These and similar undertones are highlighted vociferously in sizeable literature. See, for e.g., David Bollier, The power of Openness: Why Citizens, Education, Government and Business Should Care About the Coming Revolution in Open Source Code Software (Berkman Center for Internet & Society Working Paper) available from: http://h20project.law.harvard.edu/opensource/h20/last accessed on August, 20 2011. See also Pekka Himanen, The hacker ethic and the spirit of the information age (2001); L. Markus, B. Manville, and C. Agnes What makes a virtual organization work? SLOAN MGMT. REV. 13.
This paper attempts to shed light on the basic conundrum raised by the success of open source software development as to why thousands of top-quality programmers contribute freely to the provision of a public good. Based on a review of the open source phenomenon and the motives behind participating in the open source development, this article adapts a signalling model in the tradition of Michael Spence\textsuperscript{91}, Lerner and Tirole\textsuperscript{92}, Leppämäki and Mustonen\textsuperscript{93} and Samuel Lee et al\textsuperscript{94}. In particular, an attempt has been made in the form of an extensive game to show that high ability programmers have incentives to participate in open source projects to produce complex products and thereby differentiate their skills from less skilled programmers. This implies that most of the high ability open source programmers are engaged in producing products to the more sophisticated technological elite, whereas less skilled closed source programmers are producing commercial products or features which typically concentrate on the most ignorant users, which inevitably forms a large portion of the general public. Therefore from the welfare paradigm the mutual existence of commercial software and open source companies becomes obvious. We can also see from our study the co-existence of high ability and less skilled programmers is also evolutionarily stable action.

Some questions still need to be addressed. Given that (a) high ability programmers have incentive to participate in open source, and (b) the action of becoming either an open source programmer or closed source programmer is evolutionarily stable, then why would some high ability programmers from few countries still choose to join the closed source structure. Perhaps this can be answered by the very culture of the societies that programmers come from.\textsuperscript{95} The manner in which cultural factors thwart the incentive structure of the programmers who are engaged in open source movement also needs some attention, but we leave it for future.

\textsuperscript{92}See Lerner and Tirole supra note 29 at 212.
\textsuperscript{93}See Leppämäki & Mustonen, supra note 71 at 20.
\textsuperscript{94}See supra note 81 at 23.
\textsuperscript{95}For a good guide to begin excavating cultural interventions, see generally, GERT HOOFSTEDE, CULTURE’S CONSEQUENCES: COMPARING VALUES, BEHAVIOURS, INSTITUTIONS AND ORGANISATIONS ACROSS NATIONS (2001).