Indian Journal of Law and Technology

Volume 12 | Issue 2 Article 3

2016

Free and Open Source Software and the Twin Tragedies

Ananth Padmanabhan

Recommended Citation

Padmanabhan, Ananth (2016) "Free and Open Source Software and the Twin Tragedies," *Indian Journal of Law and Technology*: Vol. 12: Iss. 2, Article 3.
DOI: 10.55496/YPBA2420
Available at: https://repository.nls.ac.in/ijlt/vol12/iss2/3

Free and Open Source Software and the Twin Tragedies

Ananth Padmanabhan[†]

> *Who can afford to do professional work for nothing? What hobbyist can put 3-man years into programming, finding all bugs, documenting his product and distribute for free? The fact is, no one besides us has invested a lot of money in hobby software . . . there is very little incentive to make this software available to hobbyists. Most directly, the thing you do is theft . . . Nothing would please me more than being able to hire ten programmers and deluge the hobby market with good software.*
>
> —Bill Gates' open letter in 1976 to the Homebrew Computer Club[1]

I. Introduction

The past decade has seen the rise and rise of free and open source software (FOSS) as it has made considerable inroads into Government, healthcare, media, automotive, energy, finance, aerospace, retail and several other sectors.[2] There are studies that reveal exponential expansion of the total amount of source code as well as the total number of open source projects over a period of more than ten years.[3] FOSS is arguably preferred as server software, operating systems embedded in products ranging from mobile phones to video recording devices, and scripting language for the internet, and even Microsoft had released two substantial blocks of code under this license

---

† Fellow, Carnegie India.

1 Samir Chopra & Scott D. Dexter, Decoding Liberation: The Promise of Free and Open Source Software 12 (2008).

2 *Future of Open Source Survey*, BlackduckSoftware.com, http://www.blackducksoftware.com/future-of-open-source (last visited Jan. 7, 2014).

3 Amit Deshpande & Dirk Riehle, *The Total Growth of Open Source*, *in* Open Source Development, Communities and Quality 197, 197 (Barbara Russo et al. eds., 2008).

in 2009.[4] All of this compels us to question Mr. Gates and ask whether he went terribly wrong with the grim assessment of "rational" coder behavior made way back in 1976? As a necessary corollary, what does the success of FOSS teach us, and how best can these teachings help us craft sound legal and economic policies that foster innovation in the world of technology? This paper attempts to understand the incentive structures, the hierarchical organisation, and the shared norms, all of which have certainly contributed to the growth of FOSS to where it stands today.

At the same time, this paper is no roving enquiry into the FOSS movement. There is a conceptual anchor to this study, that being the twin tragedies in property theory – the jurist's dream, the policy maker's nightmare. The first is the tragedy of the commons, and the second, its mirror image, the anti-commons. They make it important for property regimes to walk the tightrope between two extremes: i) too many beneficiaries and no effective bearers of rights, and ii) too many rights holders and diminished benefits over time. Because they find their origins in *real* property, some recalibration is required before applying them to the world of ideas and innovation. This paper does that, and then enquires whether: i) the FOSS model/s of innovation manage to successfully walk this tightrope and balance several competing interests and concerns, and ii) if it does, whether similar models can be developed by businesses to foster innovation in other realms of technology, and should be encouraged by lawmakers including by way of reform to the intellectual property laws as we have traditionally used and understood them. Academic literature that equates the FOSS model to a "commons" of ideas abound, but as discussed in Part II, a large part of such literature ignores, or deals cursorily with, why the FOSS commons has stayed clear of Hardin's tragedy.[5] This paper attempts to provide an explanation, largely highlighting the oversight of theorists who ignore nuances of the FOSS model when conveniently equating it with a plain "commons". The deeper contribution of this paper is however in the realm of the anti-commons. After recalibrating the scope of this tragedy in the intellectual property space, this paper argues in Part III that the FOSS model is a larger success when it comes to averting an anticommons tragedy than in creating a plain "commons". This success, not unexplored in much detail in the existing literature, can potentially mould the foundation for applying the open source model as a more general template while crafting policy in the intellectual property space.

---

[4] Josh Lerner & Mark Schankerman, Comingled Code: Open Source and Economic Development 1-2 (2010).

[5] Garrett Hardin, *The Tragedy of the Commons*, 162 (3859) Science 1243, (1968).

Part I of this paper surveys the FOSS movement from its early days to the present, including its fundamental philosophy and on-the-ground execution of such philosophy, its growth in size and scale, and the nuances in the ideological postures of its advocates. This part also looks into the different factors that have contributed to the growth of FOSS, such as the motivations of coders to be part of a "free" creative endeavor, the relatively flexible hierarchical structures that give considerable space for "free play", and the shared values that integrate coders completely into the movement thus nurturing new creative activity from time to time. Part II switches gears to property theory, focusing on the tragedy of the commons. After outlining the contours of this concept as initially developed in the context of real property, I proceed to gauge its possibility in the intangible property space. Then, I go on to explore reasons as to why this tragedy has not apparently occurred in the FOSS experience and whether such apparent non-occurrence should embolden us to discount the role of intellectual property rights in incentivizing innovation. I present an alternate vision of the FOSS model, one where both proprietary and FOSS models have in fact been instrumental to a considerable extent in averting a "theft economy" in the software world and thus preventing the occurrence of this tragedy. Part III proceeds to examine the "anticommons" tragedy, recalibrates it in the context of intangible intellectual property, and builds a case for my conclusion that the FOSS model has been effective in averting a serious anticommons tragedy in the software sector – an effect that has not been given its due recognition. A short conclusion follows in Part IV, where I discuss, in brief, the lessons for copyright and patent policy from this success of the FOSS movement in averting an anticommons problem.

II. The Land of the "Free"

The one revolutionary idea in FOSS which can be considered the acorn for the oak tree is the decision to provide end users with the source code. Everything else, including the flexibility in fixing bugs and cleaning up or even quickly replacing flawed versions, the motivation of several young coders to write software and contribute to the "movement", the more democratic and less rigid structures of hierarchy in open source software project, and the comfort that industries today have in using open source software, can be perceived as direct or indirect consequences of this decision. In sharp contrast, proprietary software only allows the end user a limited license to use the software, with no access to the source code. The source code is protected as a trade secret, and redistribution of the software is impermissible

under copyright law. This striking difference between FOSS and proprietary software in their approaches to the disclosure or otherwise of the source code forms the fulcrum of the study here. For this reason, the story behind the free software movement, which is also interlinked in interesting ways with the history of coding and software in general, deserves brief narration.

A. (R)evolutionary Days

The Electronic Discrete Variable Automatic Computer (EDVAC), designed and delivered for use in 1949, marked a seminal moment in the world of machines. The world had its first prominent "stored program" machine – one with which humans could interact purely through software without changing or rewiring the hardware to suit new functions and data.[6] But this was simply not enough because the world required high-level programming languages so that programs written in them could be automatically translated by compilers into machine code. There was real urgency in moving beyond "assembly languages" and batch processing. The concerns of innovation in this field and era were multifarious and diverse, and only collaborative efforts could tackle them. The Project for the Advancement of Coding Techniques (PACT), a collaborative venture involving IBM and four of its customers, worked on writing compiler software. The Society to Help Alleviate Redundant Effort (SHARE) worked in collaboration to write library routines that all of its members could use. These collaborative efforts were good for the industry too, as IBM saw increased acceptance of its equipment.[7] The Digital Equipment Corporation (DEC) expressly encouraged customers to participate in the ongoing development of its products because it simply lacked the internal resources to develop software. DEC went to the extent of furnishing customers with copies of its technical manuals. This openness and flexibility attracted universities to use its machine, the Programmed Data Processor (PDP), leading to a hacker culture built around the machine.[8] Information sharing, much needed for any effective collaboration, became the name of the game. The world of software development simply could not afford wastage of time spent reinventing the wheel due to lack of coordination.[9]

In 1969, UNIX came. UNIX, the first operating system to be written in C – a machine-independent high level programming language – tantalizingly promised a bridge between 'geeks' working on different hardware platforms.

---

[6] Samir Chopra & Scott D. Dexter, *supra* note 1, at 3-4.
[7] *Id*. at 6.
[8] *Id*. at 8.
[9] Steven Levy, Hackers 28-29 (2d ed. 2010).

AT&T, UNIX's "owner", could not enter computing business due to antitrust barriers, resulting in free licensing of UNIX *along with the source code*. UNIX was also the first operating system to include core internet software like the TCP/IP networking protocols. This provided substantial nourishment to the communication culture, fueled by e-mail and bulletin boards that had sprung up around the ARPANET and its backbone PDP-10 sites.[10] The hobbyists and enthusiasts who were, around the same time, experimenting with hardware to lay the foundation for personal computing, strongly believed that software must be free and open in order to spread the word about the growing power of personal computing. Thus, by 1975, three separate hacker cultures were thriving: the ITS community at MIT, the UNIX/C networked crowd, and the personal computing enthusiasts located largely on the West Coast. In these communities, the notion of software as a good that could be sold, or as property that could be stolen, was alien.[11]

But this notion was increasingly gaining purchase in the fledgling computer industry, and soon enough, the hacker space too. By the mid-1980s, a new criterion for hacker stardom had crept into the equation, in addition to elegance, innovation, and coding pyrotechnics: awesome sales figures.[12] DEC weakened the ITS community by discontinuing its PDP-10 series, and Symbolics, a spin-off company, depopulated it by hiring many of its hackers. The modern computers of this period, such as the VAX or the 68020, had their own operating systems, but one had to sign a non-disclosure agreement to procure even the executable copy, let alone the source code.[13] By 1983, the hundreds of proprietary software licenses riding on the microprocessor wave had become strong enough to satisfy courts and deter potential infringers.[14] Once the antitrust barrier against AT&T had run its course, they started licensing UNIX on proprietary terms and stopped providing access to the source code even for academic purposes.[15] Licensees of UNIX, including IBM, Sun and Microsoft, developed their versions of UNIX and licensed these versions only in object code form, resulting in "forking", i.e. the development of many incompatible versions.[16]

As Richard Stallman recounts, these and other developments that tilted the balance heavily in favor of proprietary software models, presented him

---

[10] Samir Chopra & Scott D. Dexter, *supra* note 1, at 10.
[11] *Id*. at 10-11.
[12] Steven Levy, *supra* note 9, at 389.
[13] Richard M. Stallman, Free Software, Free Society 7-8 (2d ed. 2010).
[14] Sam Williams, Free as in Freedom 2.0: Richard Stallman's Crusade for Free Software 99 (2d ed. 2010).
[15] Samir Chopra & Scott D. Dexter, *supra* note 1, at 136-37.
[16] Heather J. Meeker, The Open Source Alternative: Understanding Risks and Leveraging Opportunities 5 (2008).

with a "stark moral choice" to join the "proprietary software" club, leave the field of computers, or write a program that made a free community possible *once again*.[17] It was his decision to go with the third option that led to the *re-birth* or *revival* of free software. In January 1984, he quit his job at MIT and began writing GNU software. When GNU Emacs, the text editor written by Stallman as an alternative to the proprietary Gosling Emacs, grew in popularity, other coders started getting involved with the GNU Project. This necessitated more funding, and the Free Software Foundation (FSF) was born as a consequence.[18]

Though the initial goal of the GNU project was to develop the complete GNU operating system and then release it, the voluntary nature of the code-writing activity resulted in users spending unequal amounts of time on different components of the system. The self-allocation of activity depended in large measure on the popularity of each component, and the inclination of coders to keep perfecting existing components without writing new ones. While this process made these programs much more powerful, and attracted both funds and contributors to the GNU Project, it also delayed completion of a minimal working system by several years.[19] By 1990, the only major missing component was the kernel. The GNU Hurd, which was being internally developed as the kernel, failed to live up to the mark. However, at that point, Linus Torvalds who had developed Linux – a Unix-compatible kernel – made it available as free software. This led to integrating Linux with the GNU system and giving the world its first free and complete operating system, GNU/Linux, in 1992.[20] From that point, the free software movement has indeed grown in an unprecedented manner, the important reasons for which shall be discussed in the following sub-parts.

The above narrative, though brief, becomes important to understand the political economy behind the free software movement.[21] The key fact that merits highlighting here is the comprehension that there existed a culture of information-sharing and collaboration in the early days of the software industry. "Free" was as much a norm as "proprietary" before proprietary took the lead at just the point in time when it was technically becoming more feasible to share and collaborate through an emerging internet.[22] The existence of a parallel free culture was definitely instrumental in prompting

---

[17] Richard M. Stallman, *supra* note 13, at 9.
[18] *Id*. at 11, 13.
[19] *Id*. at 17.
[20] *Id*. at 19.
[21] Samir Chopra & Scott D. Dexter, *supra* note 1, at 1.
[22] Karl Fogel, Producing Open Source Software: How to run a Successful Free Software Project 13 (2009).

developers like Stallman to resist losing the battle to proprietary, and to try and work out innovation models within a free system.[23] In other words, the battle was more about regaining free culture than creating it anew.

This distinction becomes important in Part II, when examining whether, and if so to what extent, the free software model can be considered a template for commons-based production.

B. Shared Norms and Hereditary Licenses

The GNU Project may have been the vision of a few determined coders, but the free software movement has moved on to achieve much larger and grander scale and presence today. The foundation of shared norms and values provided early on and around which all software development was intended to take effect, is an important reason for this growth. Indeed, if Stallman or Linus Torvalds were only feeling charitable, they could have dedicated GNU/Linux to the public and thus given it for "free".[24] But that was not the concept of freedom that Stallman or FSF believed in, as revealed through the recursive "copyleft" mechanism that they chose instead. Theirs was a notion of freedom influenced by the practice of working with source code and the benefits this brought about to both end-user experience and continued innovation. The Free Software Definition, in pursuance of this notion, lists as integral to any "free" software, four freedoms, namely:

(i) to *run* the program, for *any* purpose,

(ii) to *study* how the program works, and to *adapt* it to one's needs,

(iii) to *redistribute* copies, and

(iv) to *improve* the program and *release* such improvements to the public.[25] Without open access to the source code, the second and fourth freedoms cannot be meaningfully exercised. To guarantee the same user freedoms over improvements and derivative adaptations of the original code made by other users, the hereditary licensing model came in.

---

[23] Richard Stallman, *The GNU Operating System and the Free Software Movement*, *in* Open Sources: Voices from the Open Source Revolution 53, 55-56 (Chris DiBona et al. eds., 1999). For more factual information on Stallman's tussles with the non-disclosure of source code in the period prior to the launch of the GNU Project, see Sam Williams, *supra* note 14, at 1-11.

[24] Tim Berners-Lee had done exactly that when he declared his protocol for the internet and its implementations to be in the public domain. *See* Christopher M. Kelty, Two Bits: The Cultural Significance of Free Software 103 (2008).

[25] Heather J. Meeker, *supra* note 16, at 21.

The idea behind hereditary licensing is simple enough, and in essence the same as Stallman's motivation for the GNU Project: do not to others what you would not like seeing done to yourself.[26] The GNU General Public License (GPL) which fully internalizes this motto is worth a quick study. The preamble to this License makes it clear that the "free" in "free software" has only to do with "freedom" and not "price". Even when copies of a program are distributed for a fee by the original licensee, the objective is to ensure that the recipients receive or can demand the source code and do enjoy all the rights that the original licensee had.[27] To guarantee this, GPL imposes a mirror-image restriction on the licensee: while the licensee has free access to the source code of the licensed work, any derivative work created by the licensee has to be distributed in source code subject to the same freedoms, limitations and restrictions as the licensed work. Because this is so, the subsequent licensee of the derivative work will in turn have to permit further licensees to enjoy identical freedoms in respect of modifications made by the subsequent licensee, thus spreading these freedoms in a "viral" fashion.[28] The fallout of these provisions is two-fold: (i) it prevents distribution of any covered work, i.e. the unmodified program or a work based on the program, on terms inconsistent with the freedoms and restrictions in GPL, and (ii) it effectively prevents combining software covered under GPL with those covered by other licenses that impose a different set of restrictions, most notably proprietary software and even software covered under other, less free, open-source licenses.[29] To overcome possible arguments of absence of contractual privity that subsequent licensees in the distribution chain may take up, the license also provides that whenever a covered work is conveyed, the recipient shall automatically receive a license from the original licensors, to run, modify and propagate that work, subject to the terms of the GPL.[30] The preamble and the terms of the license also protect against instances where the redistributors may obtain software patents, by making it clear that any such patents are to be mandatorily licensed along with the code.[31] The disclaimer of warranties in respect of the licensed work is coupled with an express permission to licensees to offer support or warranty protection for a fee, thus providing room for a business model.[32]

---

[26] Richard Stallman, *supra* note 13, at 9.

[27] Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing 36-37 (2004).

[28] Sections 4 and 5 of the GPL Ver. 3, available at http://www.gnu.org/licenses/gpl.html (last visited Jan. 6, 2014).

[29] Andrew M. St. Laurent, *supra* note 27, at 157.

[30] Andrew M. St. Laurent, *supra* note 27, at 43; Section 10 of the GPL Ver. 3, *supra* note 28.

[31] Section 11 of the GPL Ver. 3, *supra* note 28.

[32] Andrew M. St. Laurent, *supra* note 27, at 38; Section 4 of the GPL Ver. 3, *supra* note 28.

The GPL model does not however give out the full story. In reality, there is no one hacker ideology, but many such ideologies built on differing perspectives on free software as an end in itself, and the hostility to or acceptance of commercial software and its vendors.[33] These ideologies clashed in full public view in the mid '90s, primarily due to Linus Torvalds' increasing popularity among young coders who did not find commercial software reprehensible or unethical. Many of them, including Torvalds, grew up in a world of proprietary software, and contributed to free software without perceiving any injustice in non-free software. Therefore, they were mostly concerned about the technical inferiority of any program, not its licensing model.[34] This pragmatism and openness to commercial software soon found takers in industry as well, as best revealed by the Netscape story.[35] The Netscape episode culminated in the pragmatist camp and industry supporters replacing "free" software with the more benign expression, "open source".[36]

The diverse ideologies thriving in the "open" since the mid '90s have in turn found expression in the FOSS licensing model too. The very zealous and openly anti-commercial FSF's attitude is reflected in GPL.[37] Similarly, the pragmatist philosophy is reflected in the most permissive academic licenses, such as the Berkeley Software Distribution (BSD) license,[38] and the middle-of-the-road ones such as the Apache License that permit original modifications by contributors to be distributed on proprietary terms.[39] So, there is no singular standard for an open source license, only a spectrum of permissive licenses with differing attitudes to both free and commercial software. At the same time, the Open Source Definition (OSD) instils fundamental values into the licensing model by prescribing ten essential pre-requisites that any license must comply with in order to qualify as an "open source" license.[40] These ten principles reflect an interesting balance between retaining the core values of the free/open source movement such as distribution of the source

---

[33] Eric S. Raymond, The Cathedral and The Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary 67-68 (2d ed. 2001).

[34] Sam Williams, *supra* note 14, at 161-62.

[35] Netscape, when faced with losing market share for its Netscape Communicator, an internet browser directly competing with Microsoft's Internet Explorer, chose to shift from a proprietary to an open source model in January 1998. However, they shunned the GPL license due to various reasons, instead crafting the Mozilla Public License that attempted to tread a pragmatist middle path that would work in a "corporate" setting. *See* LAWRENCE ROSEN, OPEN SOURCE LICENSING: SOFTWARE FREEDOM AND INTELLECTUAL PROPERTY LAW 141-42 (2004).

[36] Sam Williams, *supra* note 14, at 165-68.

[37] Eric S. Raymond, *supra* note 33, at 69.

[38] *Id*. at 70.

[39] Section 4 of the Apache License, available at http://www.apache.org/licenses/LICENSE-2.0.html (last visited Jan. 6, 2014).

[40] LAWRENCE ROSEN, *supra* note 35, at 4-6.

code and unhindered re-distribution without any discrimination, and providing room for industry to come out with derivative adaptations and modifications that can be licensed on proprietary terms if so desired.

The free/open source movements do not restrict their methods to protect and propagate core values, to licensing models. There is an equally significant, much less visible, process of initiation to the world of free/open source programming that imparts the core values and beliefs of the movements to fresh coders. This process is important to ensure continued involvement of existing coders and induction of new ones. Without this process, the act of building on to code would come to a standstill, effectively freezing any free/open source project that is not backed by the industry. Again, the shared norms will necessarily vary within projects, depending on the dominant beliefs of the coders who matter. Some may be radical in their outlook, lending that perspective to new entrants and the project as a whole. Less rigid views, focused more on the technical than the political, may prevail in other cases. Regardless, there are baseline norms, and the success of these norms in welcoming, and binding, participants to the project can eventually determine its fate. In other words, by allowing users to become co-developers or contributors, and retaining their skill sets, FOSS encourages "natural product evolution". Such "natural product evolution" takes place within the norms of a community, the norms themselves being dependent on the nature of the project at hand.[41] Communities evolve in the free software world through role transformation, as community members who change their roles – such as from peripheral to active developers, for instance – also manage to change the social dynamics and reshape the structure of the community.[42]

The evolution of a FOSS community is thus determined by two factors: (i) the social mechanism of the community that encourages and enables individual role transformation, and (ii) the existence of motivated members who aspire to play roles with larger influence.[43] The motivation for members to so aspire, despite no monopoly rights over their creative output, is discussed under sub-part D, below. To explain the former, studies have relied on Legitimate Peripheral Participation (LPP), a community learning theory.[44] The scope and character of such LPP varies depending on the nature of the project, and in some sense, is inversely proportional to the "cathedral"

---

41 Yunwe Ye et al., *The Co-Evolution of Systems & Communities in Free and Open Source Software Development*, *in* Free/Open Source Software Development 61 (Stefan Koch ed., 2005).

42 *Id*. at 69.

43 *Id*. at 70.

44 *Id*. at 70-71.

structure of the project.[45] Exploration-oriented FOSS projects, which aim to push the frontier of software development collectively through the sharing of innovations, require a high level of quality that necessarily brings in a tight control over the periphery. Even service-oriented FOSS projects give lesser leeway for community evolution through peripheral participation, interested as they are in providing stable, reliable and robust services without much disruption. On the other hand, utility-oriented FOSS projects that attempt to develop functional solutions to existing problems follow a more decentralized model. Such projects provide better opportunities to peripheral developers to spread their efforts into the whole system and establish themselves as active developers or core members in the larger community.[46] Social ties such as friendship, which influence opinions and outlook in the real world, can also play an important role in the FOSS world to ensure coordinated activity, pursuit of core values, and attainment of technical results that meet a certain quality.[47] Similarly, conflicts that happen on internet relay chats and mailing lists over use of non-free tools and acceptance of newcomers into the fray do contribute to building a community of practice and strengthening teamwork.[48] Finally, sustained collaborative development of code, coupled with involvement in hacker conferences and discussions and decisions around free software licenses and project policy, reinforces ethics and belief in core values over a period of time.[49] As a result, over the course of participating in an open source project, coders develop a more vigorous and overt ethical stance toward the uniqueness of their project and the importance of free software than when first joining.[50]

C. Hierarchies and Creative Openness

The propagation of core values through licensing models and other means, as discussed above, is certainly integral to the success of the free/open source

---

45 *See* Eric S. Raymond, *supra* note 33, at 27-28, 31.

46 Yunwe Ye et al., *supra* note 41, at 73-75.

47 Thomas Basset, *Coordination and Social Structures in an Open Source Project: VideoLAN, in* Free/Open Source Software Development, *supra* note 35, at 125, 129-30, 139-42. Contrary view has been expressed in this more recent study, Chaim Fershtman & Neil Gandal, *Direct and Indirect Knowledge Spillovers: The Social Network of Open-Source Projects*, 42 The RAND J. of Econ. 70, 88 (2011). This study concludes that there is no evidence of any correlation between contributor closeness centrality and project success, though such evidence exists in the case of project closeness centrality.

48 Margaret S. Elliott & Walt Scacchi, *Free Software Development: Cooperation and Conflict in a Virtual Organizational Culture, in* Free/Open Source Software Development, *supra* note 35, at 152, 160, 162-66.

49 E. Gabriella Coleman & Benjamin Hill, *The Social Production of Ethics in Debian and Free Software Communities: Anthropological Lessons for Vocational Ethics, in* Free/Open Source Software Development, *supra* note 35, at 273, 274.

50 *Id*. at 279.

movement, but equally significant are the structures necessary to oversee what is, at first glance, a frenzied playground of coding activity. The bazaar model where "given more eyeballs, all bugs are shallow" – a less formal expression of Linus's Law: *Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone*[51] – is yet workable only when there is some form of control over the changes made by contributors to the code development process. Otherwise, Brooks's Law would have taken over, and adding more programmers to a late project would only have made it later.[52] Though the conceptual integrity of a program, in the cathedral vision of development, can only be preserved by a hierarchy with a system master architect at the very top and sub-architects below,[53] open source projects have more flexible hierarchies. This is needed to an extent to attract all those "eyeballs" in the first place. Pre-assigned division of work is likely to be perceived as repressive and thereby demotivate interested participants as well as hinder getting the best out of each person's abilities from among a scattered resource pool.[54] For this reason, coordination in free software privileges adaptability over planning, and relies on hierarchy only to resolve any tension between individual curiosity and collective coordination.[55] Such tension, if unchecked, can result in failure of the project, or its "forking" or splintering into variants.[56]

Studies of different open source projects reveal the use of central source code repositories that make use of some form of a Concurrent Versions System (CVS) to keep track of the changes made to a set of files, and to allow several developers to collaborate. Version control is principally about change management: identifying each discrete change made to the project's files, annotating with metadata including the date and author of each change, and providing this information to developers who seek it.[57] These repositories guarantee that each file change – called a commit or a "check in" – creates a new file version, thus keeping previous versions accessible as well.[58]

---

[51] Eric S. Raymond, *supra* note 33, at 30.

[52] Frederick P. Brooks, The Mythical Man-Month: Essays on Software Engineering 25 (2d ed. 1995). Interestingly, even this 20th Anniversary edition of this pioneering work makes no reference to free or open source models of software development.

[53] *Id*. at 257.

[54] Johan Soderberg, Hacking Capitalism: The Free and Open Source Software Movement 156-57 (2008).

[55] Christopher M. Kelty, *supra* note 24, at 211.

[56] Josh Lerner & Jean Tirole, *Economic Perspectives on Open Source*, *in* Perspectives on Free and Open Source Software 47, 53 (Joseph Feller et al. eds., 2005).

[57] Karl Fogel, *supra* note 22, at 48; Audris Mockus et al., *Two Case Studies of Open Source Software Development: Apache and Mozilla, in* Perspectives on Free and Open Source Software, *supra* note 56, at 163, 167.

[58] Jesper Holck & Niels Jorgensen, *Do Not Check in on Red: Control Meets Anarchy in Two Open Source Projects, in* Free/Open Source Software Development, *supra* note 35,

Because debugging is a vital process required for a program to transition to a programming product or a component in a programming system,[59] the technical infrastructure also necessarily includes a bug-tracking system. These centralized defect-tracking systems register information reports on bugs provided by any developer, and facilitate the committal of changes to the repository to address specific bugs.[60] Difficulties in the effective sharing of information form the rationale for Brooks's Law, which frowns upon the involvement of too many coders in a software development process. The Law especially relies on the sequential nature of debugging, which requires additional time to be spent on communication between the different coders.[61] To mitigate this concern, open source systems, in addition to the centralized bug-tracking system, make use of mailing lists, newsgroups and real-time chat systems, to channel the flow of communication and to bring everyone on to the same page without much delay.[62]

While all developers are allowed to download files from the repositories and work on them, only "committers", a chosen group of developers with special privileges, are allowed to commit changes to files in the repositories. Developers have to usually demonstrate their competence through high-quality contributions for a certain period, before they are considered for committer status. This decision may also require affirmance from more senior committers and supervisors. Even after being conferred committer status, certain open source projects provide mentorship avenues to newly anointed committers.[63] Open source projects such as GNOME, where different corporate partners pay their own employees to participate in code development, maintain a rigid distinction between volunteers who can contribute to different modules, and "contributors" who can actually commit these changes to the repository. In such cases, volunteers have to submit their patches for review and committal by these "contributors", who alone have access to the CVS.[64]

However, there are open source projects that follow more relaxed rules when it comes to committing changes to the repository, relying on more of an honor system. Here, developers are permitted to commit changes anywhere in the system but requested to confine their changes to certain specifically

---

at 1, 5-7.

59 Frederick P. Brooks, *supra* note 52, at 4-6.

60 Jesper Holck & Niels Jorgensen, *supra* note 58, at 7-8; Audris Mockus et al., *supra* note 57, at 168.

61 Frederick P. Brooks, *supra* note 52, at 16-19.

62 Jesper Holck & Niels Jorgensen, *supra* note 58, at 9-10.

63 *Id*. at 13-14.

64 Daniel M. German, *Software Engineering Practices in the GNOME Project*, *in* Perspectives on Free and Open Source Software, *supra* note 55, at 211, 214-15.

assigned portions. There are advantages to this relaxed approach too, such as saving on administrative overheads entailed in granting wider privileges when coders expand their skill sets, and more importantly, encouraging an atmosphere of trust and mutual respect.[65] Some open source projects also have 'super-reviewers' who review most code before it is committed,[66] and module owners who need to approve the committing of code into the particular module whose development is within their supervision.[67] In some cases, there will be different middle level managers to supervise important tasks: patch managers who make sure that every patch is followed up through to some stable state,[68] translation managers who supervise the translation of the software's documentation or the software interface itself into languages comfortable for developers from different nationalities,[69] documentation managers who ensure that the software documentation is kept organized, up-to-date, and consistent with itself,[70] issue managers who familiarize themselves with the bug tracking system and manage the tackling of duplicate, incomplete, poorly described and unaddressed issues / bug reports,[71] and Frequently Asked Questions (FAQ) managers who maintain the overall quality of FAQ and write new FAQ entries based on concerns raised in the mailing lists and other communicative media.[72]

Most successful open source projects will have significant number of people involved with the code-writing process, necessitating the presence of a top-level management to avoid "forking", to take important decisions such as the timing of new version releases, and quite simply to ensure the operational health and survivability of a project.[73] The concept of a benevolent dictator is not alien to the open source model. These individuals command respect among peers through good coding skills coupled with charm and an engaging persona that set them up for natural leadership. Their influence

---

[65] Karl Fogel, *supra* note 22, at 56; *See also* Audris Mockus et al., *supra* note 57, at 171-72. A recent study indicates that the involvement of peripheral developers becomes significant once the project matures, though contributions by a core group would be more dominant in the initial stages of code-writing. By adopting a rigid hierarchy, open source projects would expose themselves to the risk of losing out on this involvement. *See* Pankaj Setia et al., *How Peripheral Developers Contribute to Open-Source Software Development*, 23 Information Sys. Research 144, 155 (2012).

[66] Audris Mockus et al., *supra* note 57, at 191-2.

[67] Jesper Holck & Niels Jorgensen, *supra* note 58, at 13-14; Daniel M. German, *supra* note 64, at 215; *See also* Alessandro Narduzzo & Alessandro Rossi, *The Role of Modularity in Free/Open Source Software Development, in* Free/Open Source Software Development, *supra* note 35, at 84.

[68] Karl Fogel, *supra* note 22, at 148.

[69] *Id*. at 149.

[70] *Id*. at 150.

[71] *Id*. at 151; Audris Mockus et al., *supra* note 57, at 173.

[72] *Id*. at 152.

[73] *Id*. at 67.

is generally felt only as a kind of casting vote when the normal processes of discussion and deliberation fail to throw up a conclusive verdict.[74] As projects get older, they tend to move away from the benevolent dictatorship model and toward more openly democratic systems.[75] Many significant open source projects are headed by groups or core teams comparable to the board of directors in traditional organizations.[76] These groups are either nominated or democratically elected.[77]

There are complexities with the release of new versions of free / open source software that do not exist in the case of proprietary software. They arise because development is an ongoing process in open source. Not all developers may be satisfied with the existing "version", or be inclined to "stabilize" their creative efforts.[78] This is also a positive feature when viewed from a technical standpoint as the attempt to achieve perfection is not rendered immobile at any particular stage due to the release of a version. At the same time, users do not need to wait endlessly for the end product to achieve perfection to the satisfaction of all developers.[79] Open source projects manage to achieve this by following a trunk and branches model, where the trunk is the code that keeps evolving and the branches are the release versions.[80] This is easier said than done though because considerable amount of planning and decision-making goes into the timelines for release of the alpha, beta and final versions of any particular "branch".[81] Moreover, it is necessary to decide on, and communicate in a tactful manner, restrictions on changes to the development branch in a phased manner, to achieve the final release version without confusion or compromise of the overall development momentum.[82] To realize these needs, open source projects resort to a system of positive and express voting in favor of last-minute changes, or rely on a release owner who is responsible for approving or rejecting changes and bug fixes.[83] Release managers keep a track of the changes that are under consideration, already approved, or important and yet unnoticed, and prompt other developers to take timely action.[84] Companies that distribute open source products for revenue also necessarily require a team that

---

74 *Id*. at 68.
75 *Id*. at 69.
76 Daniel M. German, *supra* note 64, at 219.
77 Jesper Holck & Niels Jorgensen, *supra* note 58, at 12.
78 Niels Jorgensen, *Incremental and Decentralized Integration in FreeBSD*, *in* PERSPECTIVES ON FREE AND OPEN SOURCE SOFTWARE, *supra* note 56, at 227, 241-42.
79 KARL FOGEL, *supra* note 22, at 118.
80 Niels Jorgensen, *supra* note 77, at 242.
81 Jesper Holck & Niels Jorgensen, *supra* note 58, at 15-17.
82 KARL FOGEL, *supra* note 22, at 123, 125.
83 *Id*. at 125-26.
84 *Id*. at 127. Audris Mockus et al., *supra* note 57, at 175.

carries out economic calculations to decide on the extent of efforts that a new distribution calls for.[85]

D. Motivations, Extrinsic and Intrinsic

The success of any free / open source project, and the FOSS movement as a whole, is closely tied with the motivations of coders who are willing to devote time to a "free" endeavor. A generalized explanation offered for this phenomenon is that innovators innovate, regardless of monopolies, because the return to them from deploying new ideas is high. Free markets have never historically *guaranteed* a market to any merchant, yet innovation continues.[86] While this view may hold good in certain cases, and open source model could well be one such, not all capital investment in developing new ideas and executing them can survive absent special legal protection. Otherwise, any one purchaser could easily destroy the monopoly by reproducing the information at little or no cost.[87] Moreover, most of code-writing in open source projects is incremental, with no real opportunity for many of the individual contributors to monetize the code. They incur an opportunity cost of time, foregoing monetary compensation that could otherwise be earned by working for a commercial firm or a university.[88] However, these costs can be offset by the pleasure that coders get out of the aesthetics of writing beautiful code,[89] ego gratification from peer recognition, or even tangible benefits such as future or better job prospects, and shares in open-source based companies.[90] The complex interplay of these competing considerations of costs and benefits deserves closer scrutiny and deeper understanding, especially for the purposes of the analysis that follows in Part III of this paper.

Broadly, the motivation to involve in an open source project can be either intrinsic, such as the joy arising from partaking in the intellectual challenges

---

[85] Jesus M. Gonzalez-Barahona et al., *Analyzing the Anatomy of GNU/Linux Distributions: Methodology and Case Studies, in* Free/Open Source Software Development, *supra* note 35, at 27, 31.

[86] Lawrence Lessig, The Future of Ideas: The Fate of the Commons in a Connected World 71 (2001).

[87] Kenneth J. Arrow, *Economic Welfare and Allocation of Resources for Invention, in* The Rate and Direction of Inventive Activity: Economic and Social Factors 609, 615 (1962); Josh Lerner & Mark Schankerman, *supra* note 4, at 25.

[88] Josh Lerner & Jean Tirole, *The Economics of Technology Sharing: Open Source and Beyond* 7 (Nat'l Bureau of Econ. Research, Working Paper No. 10956, 2004).

[89] Samir Chopra & Scott D. Dexter, *supra* note 1, at 71.

[90] Josh Lerner & Jean Tirole, *supra* note 88, at 8.

of code-writing, or extrinsic, such as better jobs and career advancement.[91] Intrinsic motivation can again be separated into two components, subject to the obvious disclaimer that these components are not mutually exclusive and can inhere alongside each other in the same individual. For that matter, even intrinsic and extrinsic motivations can act in tandem to motivate a coder. These dual components of intrinsic motivation are enjoyment-based intrinsic motivation, and obligation/community-based intrinsic motivation.[92] To understand intrinsic motivation of the first kind, appreciation of the aesthetics of code-writing is essential. In this regard, parallels can be seen between descriptions of FOSS development that emphasize its spontaneous, unorganized or even chaotic qualities, and traditions in art criticism that praise the spontaneity of the creation.[93] By participating in the FOSS development process, coders place their work up for criticism by an audience of peers spread all across the world, in much the same way as artists expose the output of their creative endeavor.[94] While users are ultimately concerned only about the functionality of code, programmers have a notion of its beauty. Much as an artist extracts form from objects of experience and imposes that on canvas, the programmer imposes the form of an abstract algorithm in, and on, a particular programming language.[95] Similarly, the literary character of code introduces the possibility of a neat and clean style of writing it, thus adding a new layer or dimension of beauty. Unsurprisingly, the work of a veteran would be distinguishable from that of a novice.[96] Apart from the aesthetic quality of the output, the creative "flow" in the process of writing code and the inner joy in accomplishing a task involving intellectual challenge contribute to enjoyment-based intrinsic motivation.[97] Obligation/community-based intrinsic motivation is not different from the shared norms and values that form the foundation of the free/open source movement,[98] which have already been discussed in sub-part B above.

The most obvious extrinsic motivation is closely tied with the technical merits of the open source model, which have achieved high levels of public awareness in the past decade. Along with this success has come significant corporate presence and participation in this model, and the earmarking, by technology bellwethers, of specific portions in their budget for open

---

91 Karim R. Lakhani & Robert G. Wolf, *Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects, in* Perspectives on Free and Open Source Software, *supra* note 56, at 3.

92 *Id*. at 4.

93 Samir Chopra & Scott D. Dexter, *supra* note 1, at 74.

94 *Id*.

95 *Id*. at 77.

96 *Id*. at 78.

97 Karim R. Lakhani & Robert G. Wolf, *supra* note 91, at 4-5.

98 *Id*. at 5-6.

source contribution.[99] The reasons for enhanced corporate presence include improvement of employee skills through peer-driven training; possibility of utilizing some of the tools used for developing the open source code to address in-house technical concerns; gathering "competitive intelligence" about a competing open source project; gaining superior knowledge and understanding of the code as part of a business model that provides goods and services complementary to the open source product; and even better public relations.[100] Because of these reasons, firms that specialize in customized software and software that can be bundled with hardware are more likely to receive corporate funding than those with a focus on packaged open source software and support services.[101] Again, the flexibility offered by the open source license to create copyrightable derivative versions and modifications has an impact on funding prospects. Projects operating under BSD open source licenses are more than twice as likely to receive corporate funding as those operating under the more restrictive GPL or other licenses.[102] Apart from firm sponsorship which is a strong extrinsic motivation, coders also contribute because of delayed benefits such as career advancement and improvement of programming skills in course of time.[103] Another important signaling incentive is the ego gratification through peer recognition, which is more likely to materialize in the open source model than in the commercial software model of development.[104]

There are studies and theories which attribute greater significance to extrinsic[105] as well as intrinsic[106] motivations, and it is difficult, if not

---

[99] Josh Lerner & Mark Schankerman, *supra* note 4, at 48-50, 91-92. Some of the varied ways in which corporates contribute to open source development are by (i) formally or informally encouraging employees to spend some time contributing to open source projects, (ii) directly providing complementary services and products that are not supplied efficiently by the open source community, and (iii) permitting open source projects to make use of their proprietary code to achieve better technical results.

[100] *Id*. at 49-51, 91; Karl Fogel, *supra* note 22, at 76; Lawrence Lessig, *supra* note 86, at 69-70.

[101] *Id*. at 93.

[102] *Id*.

[103] Karim R. Lakhani & Robert G. Wolf, *supra* note 91, at 7.

[104] Josh Lerner & Jean Tirole, *Some Simple Economics of Open Source*, 50 J. of Ind. Econ. 197, 216 (2002). The primary reason for better prospects of peer recognition is due to the technical architecture of an open source project, where everyone can see for themselves and evaluate the contribution made by each developer. Because the programmer is acting independent of directions from the top, unlike in the case of a proprietary model, there is better performance attribution too.

[105] *Id*. at 217-220; Alexander Hars & Shaosong Ou, *Working for Free? Motivations for Participating in Open-Source Projects*, 6 Int'l. J. Econ. Comm. 25, 34-35.

[106] Yochai Benkler, The Wealth of Networks 60, 100 (2006); Karim R. Lakhani & Robert G. Wolf, *supra* note 91, at 7; Chong Ju Choi et al., *Global Ethics of Collective Internet Governance: Intrinsic Motivation and Open Source Software*, 90 J. Bus. Ethics 523, 524 (2009).

impossible, to conclude on this issue. Most studies rely on surveys conducted among developers, and have severe limitations, including the absence of consciousness of motives, or their deliberate or unintended suppression, on the part of developers.[107] A recent study, keeping these limitations in mind, attempts to study developer motivations by analyzing the actual contributions of developer groups rather than their stated intent.[108] The central finding of this study is that developers stand enticed by notable project features such as the openness of the license, project size, and corporate sponsorship. This is consistent with economic theory that suggests higher long-term incentives under conditions of greater visibility of performance to the relevant audience, higher impact of individual effort on the final outcome, and grander information about one's talents that the performance is structured to give out.[109] Empirical patterns from this study also reveal a greater role for intrinsic factors and reputational factors, and lesser role for other extrinsic factors such as the expectation of reciprocal contributions from users, in motivating contribution from volunteers.[110] The only reasonable position to take, based on these studies and theories, is that the motivations for coders are as varied and diverse[111] as it was for a Richard Stallman, on the one hand, and a Linus Torvalds, on the other,[112] to come together to develop GNU/Linux. At the same time, open source projects have to carefully balance intrinsic and extrinsic motivations, particularly when there is corporate involvement. Corporate sponsors may naturally expect, and assert, more influence and say in the development process, and this in turn can give rise to an apprehension of control, especially in the minds of peripheral developers. This is a recipe for disaster, as it can lead to splintering of the developer community and the "out-group" developers switching their attention to projects that are perceived as more meritocratic and less susceptible to monetary influences.[113] Peripheral developers have key roles to play both in product

---

[107] Rishab Aiyer Ghosh, *Understanding Free Software Developers: Findings from the FLOSS Study*, *in* Perspectives on Free and Open Source Software, *supra* note 56, at 23, 39.

[108] Sharon Belenzon & Mark Schankerman, *Motivation and Sorting in Open Source Software Innovation*, (unpublished manuscript, Nov 2012), available online at https://faculty.fuqua.duke.edu/~sb135/bio/Belenzon%20Schankerman%20OSS%20July%202012.pdf (last visited Jan. 10, 2014).

[109] Josh Lerner & Jean Tirole, *supra* note 88, at 8.

[110] Sharon Belenzon & Mark Schankerman, *supra* note 108, at 32-33.

[111] Fadi P. Deek & James A.M. McHugh, OPen Source: Technology and Policy 164-66 (2007); Ron Goldman & Richard P. Gabriel, Innovation Happens Elsewhere: Open Source as a Business Strategy 72 (2005).

[112] David M. Berry, Copy, Rip, Burn: The Politics of Copyleft and Open Source 116 (2008). Torvalds came to know of the free software movement because he was part of the student audience addressed by Stallman in Finland. He later recalled that although the political and ethical call to arms did not really inspire him, he saw the underlying technical logic: no programmer can write error-free code all by himself.

[113] Karl Fogel, *supra* note 22, at 75.

diffusion through awareness, and enhancement of product quality mainly in the mature stages of code development.[114] It would be unwise to jeopardize this contribution by sending out the visual of a monolithic corporate presence,[115] by swamping the field with too many extrinsic incentives that end up "crowding-out" initially dominant intrinsic motivations,[116] or by festering coordination failures that studies have shown to exist between corporate and voluntary developers.[117]

III. The FOSS Model – A Commons Sans the Tragedy?

A. The Tragedy of the Commons

The aim in Part I was to provide a landscape of the FOSS model(s), and some of the structural and philosophical reasons for their viable and stable growth. The sharing of source code, all important in most FOSS licenses, has certainly given coders a common pool of creative ideas and expression from which they stand to benefit. It is tempting therefore to visualize FOSS as a "commons"[118] – a virtual community of like-minded individuals who band together to create and share common public goods deemed important to the community.[119] In the specific context of FOSS, they are highly skilled individuals who have remarkable technical ability, are often young, keen to impress with their problem-solving approach, drawing directly on notions of meritocracy, and generally, believers in the project of science and rationality.[120] But apart from this, their division across the lines of "free" vs. "open source" models is not split into binary but set along a continuum of beliefs including one of absolute indifference to the differing ideologies.[121]

---

[114] Pankaj Setia et al., *supra* note 65, at 157-59.

[115] Karl Fogel, *supra* note 22, at 78.

[116] Stephan Meier, *A Survey of Economic Theories and Field Evidence on Pro-Social Behavior, in* Economics and Psychology: A Promising New Cross-Disciplinary Field 51, 67-68 (Bruno S. Frey & Alois Stutzer eds., 2007).

[117] Jan Eilhard & Yann Meniere, *A Look Inside the Forge: Developer Productivity and Spillovers in Open Source Projects* 1, 24 (Working Paper, 2009), *available at* http://papers.ssrn.com/sol3/papers.cfm?abstract id=1316772.

[118] Lawrence Lessig, Code and Other Laws of Cyberspace 104-08 (1999); Lawrence Lessig, *supra* note 86, at 55-72.

[119] Charlotte Hess & Elinor Ostrom, *Ideas, Artifacts, and Facilities: Information as a Common-Pool Resource*, 66-SPG Law & Contemp. Probs. 111, 120-121 (2003); Allen K. Yu, *Enhancing Legal Aid Access Through An Open Source Commons Model*, 20 Harv. J.L. & Tech. 373, 374-75, 378-79 (2007).

[120] David M. Berry, *supra* note 112, at 138.

[121] *Id*. at 141-42.

The extension of this imagination of a "commons" is a world of "commons-based" peer production that relies on human cooperation rather than candies or the cane to replenish the commons.[122] In an approach that seriously questions the "rational-actor" model, FOSS is used as an example of selfless cooperation even where there are strong commercial benefits to acting in a self-interested direction. In other words, coders are contributing time and effort for free because it enhances their sense of identity and community and because the activity itself is fun. And even more surprisingly, they continue to voluntarily contribute despite other contributors getting paid, without feeling crowded-out.[123] The policy prescriptions that follow from this belief in human capacity and motivation for selfless action (or at least action less guided by extrinsic than by intrinsic factors) – well exemplified by the FOSS success story – varies according to the nature and strength of such belief, and the conditions under which the believers consider this "unusual" human behaviour to take wings. Thus, they cover the entire gamut ranging from using licenses as a tool to influence a recursive public to continue doing the "right" thing,[124] reducing copyright term for software to a five year period that is renewable by another five years, with mandatory access to the source code once the term expires,[125] granting tax benefits to donors of intellectual property who add to the creative commons,[126] ensuring that both technological protection measures and private contracting do not exceed the reach of copyright law itself,[127] permitting reproduction for noncommercial purposes and recasting copyright as an exclusive right of commercial exploitation,[128] suitably redesigning laws to reflect the shift from a pure incentive-based approach to one guided equally by intrinsic motivations,[129] and at the very least, refraining from policy measures such as expansionism of intellectual property rights, which foreclose the possibility of a commons built on intrinsic motivation.[130]

But it is precisely when scouting sound policy prescriptions that we are also compelled to look to the tragedy of the commons as a cautionary tale. The case for such a tragedy was convincingly put forth for the first time

---

[122] Yochai Benkler, The Penguin and the Leviathan: How Cooperation Triumphs over Self-Interest 1 (2011).
[123] *Id*. at 169, 178-80.
[124] Christopher M. Kelty, *supra* note 24, at 299-300.
[125] Lawrence Lessig, *supra* note 86, at 253.
[126] *Id*. at 254.
[127] *Id*. at 256-57.
[128] Jessica Litman, Digital Copyright 180 (2001).
[129] Yochai Benkler, *supra* note 122, at 241.
[130] James Boyle, *The Second Enclosure Movement And The Construction Of The Public Domain*, 66-SPG Law & Contemp. Probs. 33, 48-49 (2003).

in 1968,[131] in an article dealing with population explosion whose zone of intellectual influence has expanded well beyond this seemingly narrow, yet important, concern. The imagery used was a pasture open to all, which for years did not face excessive utilization because the herdsmen and cattle reliant on it never went above the carrying capacity of the land due to high mortality rates. At a certain point in time, population stabilizes. Now, those free to exploit the pasture numerically exceed its carrying capacity, giving rise to the tragedy. Every herdsman, being a rational actor, looks to expand his herd because the benefits of such expansion are individualized but the costs are collectively borne by all herdsmen who use the pasture. In the absence of any restrictions that place the cost of using the pasture at the doorstep of the herdsman who expands his herd, all of them hurtle to ruin.[132] This scenario, where use of the pasture is rivalrous in the sense of one man's exploitation exhausting another's capacity to put the same resource to use, has been sought to be distinguished from the world of intangible property where use of ideas is non-rivalrous because the originator of the idea is still free to use it along with every other person who receives the idea.[133] However, in the intangible property space, overuse is not the tragedy we worry about. The concern is with a different kind of collective action problem: the problem of incentives to create the resource in the first place.[134] Without an ability to exclude others from using products of the intellect, the creator of the idea would be unable to charge for the creation. To avoid this problem of inadequate incentives to create, the law steps in and creates a limited monopoly called an intellectual property right.[135]

B. Open Source and the Imaginary Tragedy?

To counter this, the open source model is used as a beacon of innovation in the absence of monopoly incentives. The argument goes that there are diverse rewards, broadly fitting within three categories, namely, monetary, intrinsic hedonic, and social-psychological, that motivate people to act in a certain way.[136] The open source model specifically teaches us that under certain conditions, the intrinsic hedonic and socio-psychological motivations make up for the absence of monopoly incentives to still result in individual action that positively enhances these set of motivations over action that is

---

[131] Garrett Hardin, *The Tragedy of the Commons*, 162 Science 1243, 1244 (1968).
[132] *Id.*
[133] Lawrence Lessig, *supra* note 86, at 22.
[134] James Boyle, *supra* note 130, at 41-42.
[135] *Id.* at 42.
[136] Yochai Benkler, *Coase's Penguin, Or, Linux And The Nature Of The Firm*, 112 Yale L.J. 369, 426-27 (2002).

simply monetarily incentivized.[137] The sustaining conditions are as important as the concept of peer-production. The modularity of the project, being its ability to be split up into independent modules that can separately be pursued by interested participants, is important to tap into non-monetary incentives. Similarly, the higher the granularity of the project in question, being its ability to be split into smaller sizes, the more likely people are to involve themselves in working on those independent modules.[138] Independent of the minimum granularity of a project, heterogeneity in the size of the modules may add to its efficiency by allowing contributors with diverse levels of motivation to collaborate by contributing modules of different sizes, whose production therefore requires different levels of motivation.[139] But it is the final set of conditions, relating to the integration of contributions by people scattered all over, where the commons tragedy comes up as a real threat to the continuance of the project.

Before addressing the nuances of peer-based production that averts this possible tragedy at the integration stage, it is important to understand the sense in which "commons" is used because it is indeed a distinct one from the pasture visualized by Hardin.[140] Hardin's pasture is an unregulated one, with untrammeled rights of grazing to every herdsman and his herd. The information commons imagined by proponents of the open source model is more on the lines of a shared resource pool, with norms and attributes of the sharing community facilitating some kind of collective action.[141] These norms evolve, adapting to changes in technology and communities.[142] In this framework, the exclusionary right in respect of property only provides one side of the story, because access, contribution, extraction, removal, management/participation and alienation are all equally important metrics in gauging the value stakeholders derive from the digital knowledge commons.[143] In other words, Hardin's pasture involved a binary between exclusion and use, while the digital commons introduces the important variable of effective governance mechanisms, including social norms and customs, to regulate

---

[137] *Id*. at 429.

[138] *Id*. at 435. Conversely, "if the finest-grained contributions are relatively large and would require large investment of time and effort, the universe of potential contributors decreases."

[139] *Id*. at 436.

[140] Garrett Hardin, *supra* note 131.

[141] Elinor Ostrom & Charlotte Hess, *A Framework for Analyzing the Knowledge Commons, in* Understanding Knowledge as a Commons: From Theory to Practice 41, 48-50 (Charlotte Hess & Elinor Ostrom eds., 2007).

[142] *Id*. at

[143] *Id*. at 52-53.

use and contribution.[144] This distinction has been extended to argue that even private property regimes and private corporations, rightly understood, are only a "managed commons", and that a movement toward private property is a movement from a 'commons' in a physical resource to a 'commons' in the social structure of individualized resource management.[145]

Using this framework of a shared resource pool, the open source model is put forth as a success story in integrating the contributions of those who form a part of the "commons". This success is largely due to the internet, which makes it fairly easy to permit contributions from a large pool of contributors. Apart from increasing the number of eyeballs, this also leads to reducing the effects of free riding because the absolute number of contributors responding to some mix of motivations remains sufficiently large.[146] There is simultaneous acknowledgement though of factors that could upset this delicate balance and cause higher defections, leading to ruin of the peer-production model. Unilateral appropriation, either through commercialization of the common efforts of all for private benefit or even by superimposition of individual values over that of the community, can dangerously cause a wedge in the community.[147] Mechanisms such as the GPL are tailored precisely to address this concern.[148]

C. Curbing Flights of Fancy – A Realistic Assessment of Open Source

Taking a sharing regime as the starting point, some discourse has emerged which critically, and to my mind rightly, examines the illusion of the "commons".[149] The reasoning goes that the success of any sharing regime would ultimately hinge on informal reciprocity norms that sustain contribution over withdrawal despite the absence of legal norms. Therefore, rationally, the cooperation gamble can only be viable when the innovators are repeat players with sufficiently low discount rate, and a reputation-based enforcement technology exists that sufficiently rewards compliance with, and penalizes violations of, the governing reciprocity norms.[150] However, for

---

[144] Yochai Benkler, *supra* note 136, at 437; Carol Rose, *The Comedy of the Commons: Custom, Commerce and Inherently Public Property*, 53 U. Chi. L. Rev. 711, 742-44 (1986).

[145] Carol Rose, *supra* note 144, at 746-47; Charlotte Hess & Elinor Ostrom, *Ideas, Artifacts, And Facilities: Information As A Common-Pool Resource*, 66-SPG Law & Contemp. Probs. 111, 123 (2003).

[146] Yochai Benkler, *supra* note 136, at 438.

[147] *Id*. at 439-40.

[148] *Id*. at 441.

[149] Jonathan M. Barnett, *The Illusion of the Commons*, 25 Berkeley Tech. L.J. 1751 (2010).

[150] *Id*. at 1764-65, 1769.

the cooperation to be a lasting and stable proposition, rather than merely a viable one, the following features become important: (i) a small group size of participants, (ii) low capital investment, (iii) low economic value to the innovative output, and (iv) roughly equivalent innovative endowments, i.e. the capacities and talents of innovators.[151] Of these, the low capital investment required to write code is perhaps more important than the others factors, when applying this framework to the open source model. This factor clearly places the open source model on a footing separate from industries such as pharma or telecommunications where the capital investment is higher. There is some evidence that suggests failure of the open source paradigm in the biotech industry to incentivize innovation, which is consistent with this thesis.[152] It is also a significant factor to the relative stability of the FOSS model that contribution to the open source pool is by coders who are of relatively equivalent talents and capacities. There is also some empirical basis to the claim that smaller, close-knit projects have been more successful than the ones that claim to tap from the global pool of coders.[153] In any event, as seen in Part I above, there is significant attempt at cohesion of coder groups through dissemination of core values.

But the more fundamental criticism of claims regarding the avoidance of a tragic commons can be made simply by revisiting the evolution of FOSS, discussed in Part I, sub-part A. This requires us to explore Hardin's commons a little more deeply. Hardin does not start off with a tragic commons. The pasture is initially one that can accommodate competing uses. It is at a certain point, when the factors leading to population decline are brought within control, that the pasture feels the pressure of overuse and ultimately comes to ruin. In every system, even the most well designed one, there could be a certain element of free riding. This metamorphoses into a tragedy when the burden of that free riding cannot be borne by the resource any more. Viewed this way, the nascent world of software programmers could be equated with Hardin's pasture in its early years. The limited commercial significance of software and the spirit of scientific enquiry that guided research in this field prior to the '80s could accommodate a system of open and unfettered appropriation of the source code. By the early '80s, software was becoming increasingly popular as the technology of the future. The possibility of using software to power machines for personal use and to solve diverse problems confronting several unconnected industries started

---

[151] *Id*. at 1770.

[152] Lisa Mandrusiak, *Balancing Open Source Paradigms And Traditional Intellectual Property Models To Optimize Innovation*, 63 Me. L. Rev. 303, 323-24 (2010).

[153] Martin Fink, The Business and Economics of Linux and Open Source 138-57 (2003).

lending it huge commercial significance. This could well have been the trigger point when free riding on the code written by others could potentially set in motion the chain of events leading to ultimate ruin for all. It is difficult, however, to conclusively establish this because unlike Hardin's pasture, intellectual property rights for software protection already existed though by an extension of "literary works" protection in copyright law. Proprietary software could well have averted the tragedy of the commons in two significant ways: one, by denying free riders the ease of copying that they enjoyed earlier and thus incentivizing programmers who took software out of the confines of academic research labs and scaled it up to business models of the future, and two, equally important here, providing a host of intrinsic motivations to early day free software coders to develop an alternate model. The second point can be established by resort to a simple thought experiment. Let us imagine that software protection was equivalent to Hardin's pasture, and that ideas were unprotected. Even if source code were to be kept confidential, free riders could easily replicate the end product, i.e. the machine code, and thus hurtle everyone into eventual ruin. There would be nothing unique about Stallman's clarion call to fellow coders to write free software in this already free world. What made Stallman's call special to those who contributed is the possibility of creating an alternate world, one where software was free to be redistributed, adapted, modified and self-taught. A "user community" of project participants in a non-institutional setting, a phenomenon uncommon in other areas such as biotechnology or the automobile industry,[154] was formed as a response mechanism to a private property norm. The existing private property regime did something more as well. It presented Stallman and FSF with a neat mechanism called GPL to propagate their core beliefs and create a community by inversing the exclusionary right. If not for private property in software, Stallman would have been unable to do so, and the private appropriation of his code by free riders would have destroyed the free software community.

The above analysis was only to serve the limited purpose of showing how the FOSS movement, far from being a notable exception to the tragedy of the commons, is the product of a system of private property that possibly saved the day for innovation in software. This does not in any manner preclude the possibility of using open source models of innovation in appropriate ventures, under appropriate business conditions.[155] That is a separate debate,

---

[154] Lisa Mandrusiak, *supra* note 152, at 323.

[155] For more on open source innovation in other fields, *see* Alpheus Bingham & Dwayne Spradlin, The Open Innovation Marketplace: Creating Value in the Challenge Driven Enterprise (2011); Open Innovation: Researching a New Paradigm (Henry Chesborough et al. eds., 2006); Jaap Bloem , Menno van Doorn & Erik van Ommeren, Open for Business: Open Source Inspired Innovation (2007).

not within the scope of this paper. It is wrong, however, to assert that the open source model reveals why the tragedy of the commons is not a real threat. In fact, if there is one thing we can take away from the evolution of this model, it is the important role played by private property in the creation of this model in the different ways shown above.

IV. Averting Anticommons: An Invaluable Contribution of the FOSS Model

A. The Tragedy of the Anticommons

The inability of the open source model to give much policy insight into avoiding a tragedy of the commons type situation does not take away its relevance to policy making in a different context and to avoid a different kind of problem. This problem – termed the "tragedy of the anticommons" due to the inverse character of its formulation in relation to the "tragedy of the commons" – was conceptualized by Michael Heller pursuant to his study of property underuse in post-Soviet era Russia.[156] Observing the empty Moscow storefronts and the bustling street kiosks, Heller theorized that the problem was with multiple ownership or exclusionary rights in respect of the same scarce resource, resulting in lack of consensus on optimal use of the resource and a consequential blocking of use by any of the "owners."[157] Thus, while the commons tragedy resulted in overuse, the anticommons tragedy led to underuse. As a policy prescription to avoid this phenomenon, Heller proposed that it was not sufficient to create private property rights but also important to focus on placing the entire bundle of rights in one owner.[158] If necessary, governments ought to redefine and reallocate property rights to meet this end, including by way of abolishing or expropriating previously granted rights.[159] This is because once an anticommons is formed, institutions and interests coalesce around them, giving rise to deviant strategic behavior by these beneficiaries to retain the *status quo*.[160] The transaction costs for freeing up vexatious anticommons could in fact be much higher than those incurred to resolve a commons tragedy.[161]

---

[156] Michael A. Heller, *The Tragedy Of The Anticommons: Property In The Transition From Marx To Markets*, 111 Harv. L. Rev. 621 (1998).
[157] *Id*. at 639, 668-69.
[158] *Id*. at 640.
[159] *Id*. at 641.
[160] *Id*. at 659.
[161] Francesco Parisi et al., *Duality In Property: Commons And Anticommons*, 25 Int'l Rev. L. & Econ. 578, 585-86 (2005).

Subsequently, Heller has applied this notion of anticommons to the study of inefficient underuse of intellectual property rights.[162] He has also recast the anticommons tragedy on a larger canvas, using it to elucidate the problem of gridlock in free markets.[163] A major conceptual critique, perhaps due to Heller's fuzzy articulation of the anticommons in an intangible property space, has been that every piece of intellectual property is a different resource.[164] Therefore, the problem is not one of underuse of any particular resource but a general problem with effective property governance of different resources. The conceptual solution to this problem may well lie in larger collective action rather than effective bundling, and anticommons may, to this extent, obscure the real problem.[165] A large part of this critique stems from failure on Heller's part to accurately rearticulate the anticommons problem, at a conceptual level, in the intangible property space. Revisiting the commons tragedy in the intangible space would be helpful to resolve this conundrum. As we already saw, the commons tragedy in this space pertained to the specific problem of under-innovation to create future intellectual commodities. If we inverse this, the anticommons problem in the intangible property space would be all about over-incentivisation due to exclusionary rights over certain types of incremental and basic innovations. This would finally lead to a gridlock situation where the room for future innovation is severely curtailed by exclusionary rights that cover important building blocks for further research and growth. As a necessary corollary, success in overcoming the tragedy lies not in bundling these rights over incremental innovations but refusing to grant such wide-ranging exclusionary right incentives in the first place.

Some of the instances from intellectual property rights, which Heller uses to demonstrate the unhappy gridlock effect, make sense once we rearticulate the problem in this conceptual direction of over-incentivisation and blocking of optimal property use. For instance, the opening up of upstream biomedical research to *excessive or uncoordinated* creation of *private property* results in over-incentivisation,[166] and inability to conduct downstream

---

[162] Michael A. Heller & Rebecca S. Eisenberg, *Can Patents Deter Innovation? The Anticommons in Biomedical Research*, 280 Science 698 (1998).

[163] Michael Heller, The Gridlock Economy 2 (2008).

[164] David Lametti, *The Concept of the Anticommons: Useful, or Ubiquitous and Unnecessary?*, *in* Concepts of Property in Intellectual Property Law 232, 243 (Helena Howe ed., 2013).

[165] *Id*. at 244.

[166] Michael A. Heller & Rebecca S. Eisenberg, *supra* note 162, at 698 ("A researcher who may have felt entitled to coauthorship or a citation in an earlier era may now feel entitled to be a coinventor on a patent or to receive a royalty under a material transfer agreement. The result has been a spiral of overlapping patent claims in the hands of different owners, reaching ever further upstream in the course of biomedical research.")

medical innovation after a certain point in time.[167] This point specifically occurs when the transaction costs become high enough to dissuade future researchers from licensing with the multiple patent owners who have already occupied the field, or the patent applicants whose patents are pending.[168] There is no *inevitable* underuse of any particular patent in this situation but there is still a *strong possibility* that the existing patents block each other out,[169] apart from *certainly* hampering future innovation. Apart from lending conceptual strength to the clearly unfair instances of gridlock pointed out by Heller, rearticulating the anticommons problem also compels us to acknowledge that not every such instance is truly within the conceptual contours of this problem. For instance, the licensing problem that Google Book Search has run into is not a "gridlock" because of the blocking effect of anticommons. It is simply a gridlock caused by the overarching reach of this project across a wide range of copyrighted material.[170] Solutions such as collective licensing for radio stations do not again resolve an anticommons problem,[171] because no one songwriter is anyways blocking another from licensing out to a radio station, and no radio station is looking to create a new song. This can be contrasted with the truly anticommons gridlock faced by the documentary on Martin Luther King, Jr., where existing copyrighted works on the same subject effectively blocked the dissemination of a new one.[172]

Regardless, Heller's identification of causative factors leading to an anticommons tragedy in the intellectual property space is accurate. He identifies two of them: i) the creation of concurrent fragments of property rights over connected innovations in the same field of scientific research, and ii) reach-through license agreements that vest rights in the owner of a patented invention used in upstream stages of research, over subsequent downstream discoveries that rely upon such invention.[173] The second factor flows from the first in most cases because had the concurrent fragments not been created in the first place, the subsequent researcher would not have had to submit himself to a reach-through license in order to use the fragment as part of his research. Both these factors combine to create multiple interests over

---

[167] *Id*. at 699.

[168] Michael Heller, *supra* note 163, at 50-54.

[169] The airplane manufacturing gridlock, where different patent holders threatened to sue each other for working their individual patents and thus blocked the manufacture of warplanes, is a case in point. Eventually, the U.S. Government had to intervene through legislation, create a 'compulsory patent pool', and free manufacturers from the threat of crippling patent litigation. *See Id*., at 30-31.

[170] *Id*. at 29-30.

[171] *Id*. at 72, 196.

[172] *Id*. at 9-11.

[173] Michael A. Heller & Rebecca S. Eisenberg, *supra* note 162, at 699.

connected ideas that can motivate further research and innovation, thereby hiking up transaction costs for those who need to make use of all these ideas as part of their research and causing potential holdout problems.[174]

B. The FOSS Model and the Anticommons Story

A possible solution to an anticommons gridlock, emerging from the above identification of causative factors, is private action.[175] Idealistic as this may sound at first glance, such action has been forthcoming in certain situations from actors responding to this problem purely out of rational self-interest. Property-preempting investments (PPIs), where private firms spend significant sums of money to create assets that preempt intellectual property rights for strategic reasons, are gaining in popularity in industries such as biotechnology and software.[176] PPIs work on the premise that once information enters the public domain, it cannot be privatized.[177] Biotechnology and software are two areas where PPIs have been deployed with fair degree of success in turning property over to the public domain. In particular, efforts such as the Merck Gene Index, a public database of gene sequences corresponding to expressed human genes, created with the contribution of several million dollars from Merck, and the Single Nucleotide Polymorphism (SNP) consortium to place SNPs, valuable as 'disease markers', in public domain through the conjoint efforts of private firms and nonprofit research organizations, have thwarted potential anticommons effect in the biotech field.[178] The involvement of private firms, including IBM, in FOSS development, has also been explained as an attempt to preclude property rights entanglements on a key "input". Because Linux comes without the threat of leverage and dominance that are always present with a proprietary operating system, both IBM and its customers can control their own fate and rely on the GPL to commit to future or customized versions of Linux without the looming threat of high transaction costs.[179]

This is definitely a significant impact of the FOSS model in facilitating a private solution to the anticommons problem. But an even larger contribution of this model in this direction has gone unrecognized. The model itself, and not just private investments in the model, has been instrumental in averting anticommons tragedy. In Part II, while teasing out the distinction

---

[174] *Id*. at 700.
[175] Michael Heller, *supra* note 163, at 70-71.
[176] Robert P. Merges, *A New Dynamism in the Public Domain*, 71 U. Chi. L. Rev. 183, 185 (2004).
[177] *Id*. at 186.
[178] *Id*. at 188-190.
[179] *Id*. at 192-93.

between Hardin's pasture and the FOSS world, an important distinguishing factor was seen to be the origins of FOSS in a world of proprietary software. In other words, the prior existence of legal protection over this "field" made it different in its content and character from Hardin's pasture. However, there is a conceptual prism using which such "commons", carved out from a larger proprietary field, can be better understood. This is the notion of a "constructed commons" that can be used to solve innovation problems.[180] In this regard, eight clusters have been identified to investigate any particular constructed cultural commons, with the eventual goal of relating particular characteristics to the results produced by certain types of sharing arrangements.[181] These are: i) the relevant history and narrative of the commons;[182] ii) the entitlement structures and resource provisions that define its contents;[183] iii) the institutional setting and the social practices, disciplines and norms that the commons inhabits;[184] iv) the formal legal structures put in place to facilitate collective action via the commons;[185] v) governance mechanisms such as membership criteria, resource contribution and appropriation standards, decision-making rules, provisions for resolving conflicts over membership and resources, and sanctions for violations, that guide the operation of the commons;[186] vi) the interface between internal governance mechanisms (cluster no. v) on the one hand and external mechanisms (cluster nos. ii, iii and iv) on the other;[187] vii) specific solutions to innovation problems that the commons can be associated with;[188] and viii) the costs and risks associated with the commons.[189] The relevance of these clusters is not confined to delineating the contours of any "constructed commons". It extends to helping us distinguish between a "constructed commons" in the true sense of that expression, and other PPIs including "defensive publications" that render a patent application "anticipated" or at least "obvious", and thus disentitled to patent protection.[190]

Heller recognizes the role of such voluntary arrangements in overcoming gridlock, and even mentions open source as one such mechanism along with

---

[180] Brett M. Frischmann et al., *The University As Constructed Cultural Commons*, 30 Wash. U. J.L. & Pol'y 365, 266 (2009).
[181] *Id*. at 374.
[182] *Id*. at 374-75.
[183] *Id*. at 375.
[184] *Id*.
[185] *Id*. at 375-76.
[186] *Id*. at 376.
[187] *Id*.
[188] *Id*.
[189] *Id*. at 377.
[190] *See* Gideon Parchomovsky, *Publish or Perish*, 98 Mich. L. Rev. 926 (2000); Douglas Lichtman et al., *Strategic Disclosure in the Patent System*, 53 Vand. L. Rev. 2175 (2000).

his more explored one of patent pools.[191] To truly appreciate the contribution of FOSS in this direction though, a simple thought experiment, similar to the one carried on in Part II while demonstrating proprietary software's contribution in averting a commons tragedy, would suffice. As seen from the evolution and practice of programming, most code writing builds on existing code and is in that sense, derivative. Now imagine a world where Stallman had chosen not to exercise his "good conscience", not to write his legendary forum posting on 27th September, 1988 appealing to the community of coders,[192] and instead, signed the non-disclosure and software license agreements that were gradually taking over as the industry norm. Apart from its direct practical consequences such as a paid-for Android or more expensive server software, this world would have led to a race to create multiple proprietary versions of code, each of which could potentially block the other out over time. The rationale for this effect can be traced to the peculiar character of copyright protection, where the maker of the original also enjoys monopoly in respect of derivative works that adapt or modify the original. Therefore, those who create the derivative of an already licensed derivative work have to go back and take licenses from both the original coder and the owner of the first derivative work. Indeed, this would not have immediately become a concern because initial developers of derivative code could well take licenses from the original coder. Problems in licensing, including hike in transaction costs and deleterious blocking effects, would surface only once a certain threshold was crossed. This threshold would depend on the number of derivative versions in the field and the splintering of rights in such versions in the hands of different actors. Soon, the frenzied innovation activity in the evolutionary days of software would have dried up or become prohibitively expensive. It would not be a stretch of imagination to contend that Stallman's decision to keep the source code open, and more important for the purposes of the anticommons tragedy, to nullify private property in derivative versions of his code using the GPL mechanism, came in at exactly this juncture.

Unfortunately, while conceptualizing "open source", it has been categorized as a "constructed cultural commons", which arose as a solution to collective action, coordination, or transactions cost problems that existed *apart from* intellectual property rights.[193] This is in contrast to other pooling arrangements such as the SNP consortium, discussed above, and the publicly available databases of genomic sequences that are part of the Human

---

[191] Michael Heller, *supra* note 163, at 196-97.
[192] Richard M. Stallman, *supra* note 13, at 26-27.
[193] Brett M. Frischmann et al., *Constructing Commons In The Cultural Environment*, 95 Cornell L. Rev. 657, 691 (2010).

Genome Project. These have been categorized as "constructed cultural commons", which arose as a solution to collective action, coordination, or transactions cost problems that existed *because of* intellectual property rights.[194] The above bracketing of "open source" within the former category is incorrect, and it fits as much into the second category as patent pools. Analysis of the FOSS model using the eight clusters for investigating a "constructed commons", mentioned above, makes this amply clear. To do so, let us revisit Part I of this paper.

First, as shown in sub-part A of Part I, the evolution of FOSS (cluster no. i) was a response to the proprietary norms in intellectual property law generally, and copyright law and trade secrecy protection in particular (cluster no. vii). The major resource for this "constructed commons" (cluster no. ii) was the "source code" that would otherwise be kept secret, or rendered unusable, because of trade secret and copyright protection respectively. The core values of the FOSS movement, discussed in sub-part B of Part II, were built around a strong anti-intellectual property rights discourse, and the institutional setting of this community (cluster no. iii) can be contrasted with that of the proprietary software industry. The GPL licensing mechanism, discussed in the same sub-part, was craftily designed to propagate a commons of "free software" that would otherwise be monopolized by resort to copyright law and trade secret protection. This also explains the choice of copyleft (cluster no. iv) over mere dedication to the public domain. Unlike patent law, where such dedication could have preempted the patentability and monopolization of an incremental idea by a subsequent innovator, the low threshold of 'originality' required for copyright protection of derivative works in copyright law would have possibly resulted in monopolization of incremental derivative works built on works already in the public domain.[195] Apart from this, the institutional structures for internal governance (cluster no. v), discussed in sub-part C of Part I, show how different open source projects have encouraged decentralized innovation while at the same time, brought in mechanisms to control "forking" and uncoordinated innovation (cluster no. vi). When viewed in tandem with the different extrinsic and intrinsic motivations for participation in open source projects, it is clear that the focus has always been on creating an innovation model that serves as an alternate paradigm to the "incentive structures" of closed models of innovation built on intellectual property rights (again, cluster no. vii). A major risk

---

[194] *Id*. at 692.

[195] For more clarity on the low threshold of originality required for independent copyright protection for derivative works, *see Schrock* v. *Learning Curve Int'l Inc*, 586 F 3d 513 (7th Cir 2009); *L.Batlin & Son, Inc.* v. *Snyder*, 536 F 2d 486 (2d Cir 1976); *Durham Industries, Inc.* v. *Tomy Corpn*., 630 F 2d 905 (2d Cir 1980).

associated with the commons (cluster no. viii) is also caused by intellectual property rights: the possibility of a private appropriation, rendered stronger in the case of more flexible licenses such as Apache and BSD, as shown in sub-part B of Part I. For all the above reasons, the FOSS model is indeed a "constructed commons" created in response to problems caused by intellectual property rights. And it is one that has, with fair bit of success, averted an anticommons problem in the intellectual property space.

V. Concluding Remarks And Lessons for Intellectual Property Policy

The two substantive arguments made in this paper can be summarized as follows: (1) the FOSS model cannot be used to convincingly put forth a case against the occurrence of a tragic commons in a world without private property; and (2) the FOSS model can, however, be considered a successfully constructed cultural commons that has gone a long way toward preventing an anticommons problem in the software industry. The policy measures suggested below are suitably tailored to fit within what would naturally follow from these arguments. These measures are only guidelines for the direction that statutory reform may take, and not an exact articulation of the proposed reform.

The FOSS model shows that the time is ripe for the United States to expand its system of moral rights protection beyond visual artistry to other fields of creativity and innovation, such that creators are incentivized through attribution. Reputational benefit has been a strong motivation for coders to participate in FOSS development. Most FOSS licenses guarantee, through effective notice mechanism, the dissemination of author information. Even otherwise, the CVS mechanism and other technical infrastructure facilitating FOSS collaboration ensure that the coders who offer programming solutions to complex problems are duly credited for their contribution. This is not quite the case with proprietary software. By legislating for a strong moral rights protection akin to what exists in Europe or even India, the United States would be formally recognizing an intrinsic motivation that can go a long way in attracting creative contribution to collaborative efforts in various fields of innovation. This would also act as an appropriate trade-off in settings that involve incremental innovation, as compared with the alternate option of incentives through property rules.[196] The latter kind

[196] *See* Guido Calabresi & A. Douglas Melamed, *Property Rules, Liability Rules, and Inalienability: One View of the Cathedral,* 85 Harv L Rev 1089 (1972).

of incentives can result in economic inefficiency,[197] as well as difficulties in redistribution of entitlements[198] at a certain point when problems, such as the anticommons problem for instance, freeze up innovation and necessitate such redistribution.

This takes us directly to the issue of property incentives for derivative works in copyright law and incremental innovations in patent law. Models such as FOSS and patent pools teach us the perils extant in property rules that protect incremental innovations. Therefore, a suitable combination of liability rules or effective compulsory licensing mechanisms, along with the tightening of standards to attract property protection for incremental innovations, is essential to avoid a gridlock. First, copyright law must be changed to reflect the position that derivative works are entitled to copyright protection only if they meet a heightened standard of originality. Similarly, patent protection should be denied to new forms of known substances or inventions unless they meet certain well-defined and enhanced levels of efficacy. Second, even in such cases, copyright law must necessarily make a distinction between pure works of art and fiction, and academic works or computer software that are useful for purposes of further modification. In the case of the latter, there is a compelling State interest in avoiding a gridlock, and the only way in which this interest can be promoted is by either replacing property protection against derivative works with liability rules against misappropriation, or by retaining property protection but putting in place effective mechanisms for compulsory licensing of the works. Similarly, patent protection for general purpose technologies[199] and upstream research tools that can be used to create further products downstream should be coupled with an effective compulsory licensing mechanism for such inventions so that a gridlock in future innovation is avoided. Finally, trade secret protection must be denied in any situation where the innovator asserts copyright protection in respect of academic works or computer software. The innovator must thus be forced to make the trade-off between the benefits of non-disclosure and disclosure, in a manner akin to patent protection. Only this can ensure the avoidance of a gridlock that can otherwise be caused by resorting to trade secrecy and non-disclosure agreements.

---

[197] *Id*. at 1106-07.

[198] *Id*. at 1110.

[199] *See* Boyan Jovanovic & Peter L. Rousseau, *General Purpose Technologies, in* Handbook of Economic Growth Vol. IB 1182, 1184 (Philippe Aghion & Steven N. Durlauf eds., 2005).