THE PENGUIN AND THE CARTEL: RETHINKING ANTITRUST AND INNOVATION POLICY FOR THE AGE OF COMMERCIAL OPEN SOURCE

Stephen M. Maurer*

I. INTRODUCTION

The nice thing about the open source ("OS") phenomenon is that it changes faster than academics can study it. Ten years ago, most OS collaborations were organized around noncommercial, "fun" motives like altruism, hobbyist interest, and the like.[1] By contrast, today's OS projects are mostly commercial.[2] Even if our theories and policy prescriptions were right ten years ago, the ground has shifted. This Article asks how judges and policymakers should manage the new commercial OS. In the process it uncovers a paradox. On the one hand, OS lets companies share costs. This potentially gives them the power to create far more software than ever before. But sharing also has a dark side. Since all OS companies offer consumers exactly the same shared codebase, no company can offer better shared software than its rivals. The result is a *de facto* cartel that suppresses competition and incentives to invest. Strangely, then, OS is self-limiting: Companies do share, but write far less shared code than they ought to.

This Article untangles the paradox of OS sharing and asks what judges and policymakers can do to help OS reach its full potential. Part II sets the stage by describing the rise of commercial OS over the past ten years. It also profiles a leading commercial OS collaboration (The Eclipse Foundation) and describes the various design issues that face such organizations. Part III examines how companies make investment decisions in OS, closed source ("CS"), and mixed OS/CS markets. Part IV uses these ideas to analyze when OS collaborations should and should not be permitted under the Sherman Act. It argues that OS collaborations can usually write licenses that satisfy the rule of reason. It also

* © 2012 Stephen M. Maurer. University of California Berkeley School of Law and Goldman School of Public Policy. 2607 Hearst Ave. Berkeley, CA. 94720-9657, smaurer@berkeley.edu. I wish to thank Bronwyn Hall, Deven Desai, Sebastian von Engelhardt, Andreas Freytag, Michael Katz, Jenny Kwan, Prasad Krishnamurthy, Mario Madden, Steve Mutkowski, Suzanne Scotchmer, Scott Selby, Jim Spohrer, Jeffrey Thompson, and Andrew Torrance for helpful conversations. This work was supported by generous grants from the Sloan Foundation, the National Science Foundation (Grant No. SES 20082737), and Nokia Corporation. Any oversights or errors are solely due to the author.

[1] For a comprehensive review of these early theories, see Stephen M. Maurer & Suzanne Scotchmer, *Open Source Software: The New Intellectual Property Paradigm*, *in* 1 HANDBOOKS IN INFORMATION SYSTEMS: ECONOMICS AND INFORMATION SYSTEMS 285 (Terrence Hendershott ed., 2006).

[2] *See infra* notes 7–9.

proposes two safe harbors in which OS collaborations should be presumed to be procompetitive. Part V examines the antitrust status of so-called "viral" licenses. It argues that very broad licenses (notably GPL)[3] are unnecessarily restrictive and violate the Sherman Act. Part VI reviews how governments can use taxes, grants, and procurement policy to help OS sharing reach its full potential. Finally, Part VII presents a brief conclusion.

II. THE RISE AND RISE OF COMMERCIAL OPEN SOURCE

Academic understanding of OS has usually trailed the subject itself. When OS first appeared in the 1990s, scholars did little more than repeat the volunteers' own narratives. In this telling, OS was a "movement" driven by psychological ("altruism"), political ("ideology"), or postmodern incentives ("the gift economy") that defied conventional economic analysis.[4] The more scholars studied OS, however, the less strange it seemed. Indeed, many OS incentives—for example a desire for education or to demonstrate one's ability to potential employers—were transparently financial. By 2002 or so, economists had come to understand OS almost entirely in terms of familiar motives like education and signaling.[5] At the same time, commercial incentives played a relatively small role and were seldom discussed.[6]

Worldwide OS production rose slowly during the phenomenon's first decade. [Figure 1] Consider, for example, Riehle and Deshpande's leading survey of open source activity on the Internet. As late as 2002, their sample contained just 500 OS projects.[7] This changed dramatically, however, after software companies began

[3] The "General Public License" or "GPL" is one of the oldest and best-known open source licenses. It was originally written by Richard Stallman for the GNU software project. *GNU General Public License*, WIKIPEDIA, http://en.wikipedia.org/wiki/GPL (last visited Feb. 6, 2012). For further details, see *infra* text accompanying notes 174–90.

[4] For the best-known and most complete version of these arguments, see ERIC S. RAYMOND, THE CATHEDRAL AND THE BAZAAR: MUSINGS ON LINUX AND OPEN SOURCE BY AN ACCIDENTAL REVOLUTIONARY (rev. ed. 2001).

[5] *See, e.g.*, Josh Lerner & Jean Tirole, *Some Simple Economics of Open Source*, 50 J. INDUS. ECON. 197, 212–20 (2002); *see also* Maurer & Scotchmer, *supra* note 1, at 287–300 (OS communities include signaling and education as motives for creating OS products).

[6] Lerner, *supra* note 5; Maurer & Scotchmer, *supra* note 1, at 287–300. Both sources treat commercial motives as a distinctly subsidiary phenomenon.

[7] This estimate is based on a web crawler inventory of online OS code depositories. Amit Deshpande & Dirk Riehle, *The Total Growth of Open Source*, *in* PROCEEDINGS OF THE FOURTH CONFERENCE ON OPEN SOURCE SYSTEMS 197, 204 (2008), *available at* http://dirkriehle.com/wp-content/uploads/2008/03/oss-2008-total-growth-final-web.pdf. The number of OS collaborations has continued to grow exponentially since the paper was published. E-mail from Dirk Riehle, Professor for Open Source Software, Friedrich-Alexander University Erlangen-Nürnberg, Erlangen, Germany, to author (Jan. 24, 2012, 12:06 PST) (on file with author).

paying their software engineers to join OS collaborations as volunteers.[8] In December 2000, IBM announced that it would invest a spectacular $1 billion in Linux.[9] Four years later the total number of OS collaborations had grown more than fivefold. By 2006 it had doubled again to more than 4,500 projects.[10]

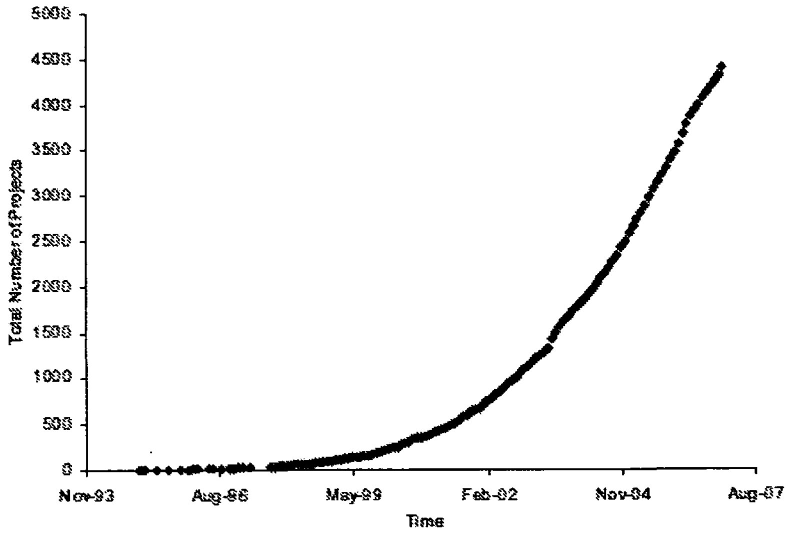

Figure 1: Number of OS Projects, 1993–2007[11]

[8] Survey researchers noticed a secular shift from hobbyist-volunteers to professionals working on company time as early as 2002. *See, e.g.*, STEFANO COMINO ET AL., FROM PLANNING TO MATURE: ON THE DETERMINANTS OF OPEN SOURCE TAKE OFF 5–6 (2007); RISHAB A. GHOSH ET AL., INT'L INST. OF INFONOMICS, FREE/LIBRE AND OPEN SOURCE SOFTWARE: SURVEY AND STUDY, PART IV, at 4 (2002), *available at* http://flossproject.org/report/FLOSS_Final4.pdf. Professor Barnett reports that 85 percent of all code that successfully enters the Linux code base originates with paid corporate employees. Jonathan M. Barnett, *The Host's Dilemma: Strategic Forfeiture in Platform Markets for Informational Goods*, 124 HARV. L. REV. 1861, 1908–09 (2011).

[9] THORSTEN WICHMANN, INT'L INST. OF INFONOMICS, FREE/LIBRE AND OPEN SOURCE SOFTWARE: SURVEY AND STUDY, PART II, 12 (2002), *available at* http://www.berlecon.de/studien/downloads/200207FLOSS_Activities.pdf.

[10] Deshpande & Riehle, *supra* note 7, at 204.

[11] *Id.*

Although exact figures are hard to come by, observers universally agree that most of the new projects are funded by for-profit companies that expect a clear dollars-and-cents return on their investment.[12] This implies that OS must be dramatically more commercial than it used to be. And indeed, industry observers estimate that commercial firms use three-quarters of today's OS software.[13] Within this category, the great majority of projects follow business plans in which OS software is bundled with some proprietary product and sold to customers.[14] This complementary good can either be a physical chattel like cell phones, a service like tech support, or closed source ("CS") software.[15]

In light of these trends, it is hardly surprising to see leading scholars argue that the old, ideology-driven image of OS needs to be rethought.[16] Yet in many ways, the new commercial OS collaborations are even stranger than the traditional "fun" ones. In the "Old Economy," companies built elaborate hierarchies to monitor and control their workers.[17] In the "New Economy," they encourage

[12] *See* Maurer & Scotchmer, *supra* note 1, at 286.

[13] Steve Lohr, *Software War Pits Oracle vs. Google*, N.Y. TIMES, Aug. 30, 2010, at B1 (remarking that days when OS was driven by sharing rather than profits "are long gone").

[14] Barnett, *supra* note 8, at 1912 n.168.

[15] There is also a second class of business models in which companies make OS software freely available to small users but charge fees to large companies. *See, e.g.*, Dirk Riehle, *The Commercial Open Source Business Model*, 15 AM CONF. ON INFO. SYS. PAPER 104, 3 (2009), *available at* http://dirkriehle.com/wp-content/uploads/2009/04/commercial_v9_revision.pdf. Here, the main goal is less about code sharing than eliciting product ideas and bug fixes from lead users. Anecdotally, this model seems to be comparatively rare and we will ignore it in what follows.

[16] For example, Professor Barnett remarks that the presence of paid employees is so pronounced that "[s]tandard characterizations of OSS development as the spontaneous coordination of . . . ideologically motivated volunteers . . . do not accurately describe at least the most successful applications in the current market." Barnett, *supra* note 8 at 1895. Similarly, Professors Lerner and Schankerman observe that, "It is the conventional wisdom among many observers, analysts, and advocates of open source software that its development of [*sic*] is driven by a widely dispersed pool of voluntary contributors. But the fact is that corporations increasingly invest large amounts of money to finance open source development, both in terms of direct finance to other companies and through paying their employees to engage in such activity." JOSH LERNER & MARK SCHANKERMAN, THE COMINGLED CODE: OPEN SOURCE AND ECONOMIC DEVELOPMENT 90 (2010).

[17] NAT'L BUREAU OF ECON. RESEARCH, 2 INNOVATION POLICY AND THE ECONOMY 8 (Adam B. Jaffe et al. eds., 2002).

employees to join self-governing OS collectives on company time.[18] Furthermore, companies do little or nothing to monitor these activities.[19]

The Eclipse Foundation. To get a better feel for commercial OS, consider the Eclipse Project. In April 1999, IBM began development of a new "Integrated Development Environment" for professional software developers.[20] Two-and-a-half years later, IBM released Eclipse 1.0 as a conventional CS software product.[21] Just one month later, however, it transferred the codebase—then valued at $40 million—to an OS "Eclipse Project" for further development.[22] Legally, the key step was adopting a "Common Public License"[23] so that users could freely use, modify and distribute the code to others.[24] However, non-IBM participation remained disappointing.[25] IBM reacted by surrendering still more control. This was done by moving the Eclipse project to an independent nonprofit corporation in February 2004.[26] At this point, large numbers of outside developers began contributing to the code base.[27] Today, Eclipse has roughly 150 member companies, and roughly one-third of these have contributed at least some software to the project since 2001.[28]

OS sharing has yielded significant value for IBM. On the one hand, non-IBM members have supplied more than half of all contributions to the project codebase

---

[18] *Id.* Companies' readiness to support paid OS volunteers can be remarkably explicit. For example, the Eclipse Foundation's rules require "Strategic Developer"-level corporate sponsors to assign at least eight full-time software developers to the project. "Strategic Consumer"-level members who contribute full-time developers pay lower dues. *Types of Membership*, ECLIPSE FOUND., http://www.eclipse.org/membership/become_a_member/membershipTypes.php (last visited Jan. 18, 2012).

[19] LERNER & SCHANKERMAN, *supra* note 16, at 41,

[20] *Eclipse Project Briefing Materials*, ECLIPSE FOUND. (2003), http://www.eclipse.org/eclipse/presentation/eclipse-slides.pdf.

[21] Sebastian Spaeth et al., *Enabling Knowledge Creation Through Outsiders: Towards a Push Model of Open Innovation*, 52 INT'L J. TECH. MGMT 411, 415 (2010).

[22] *Eclipse Project Briefing Materials*, *supra* note 20.

[23] *Common Public License - v 1.0*, ECLIPSE FOUND. (Apr. 16, 2009), http://www.eclipse.org/legal/cpl-v10.html.

[24] Users can even distribute the modified code under their own CS licenses as long as they make Eclipse's underlying source code freely available. *Eclipse Public License - v.1.0*, ECLIPSE FOUND., http://www.eclipse.org/legal/epl-v10.html (last visited Jan. 17, 2012).

[25] *See* Spaeth et al., *supra* note 21, at 419.

[26] *Id.* The new body is called the Eclipse Foundation. *See also Bylaws of Eclipse Foundation, Inc.*, ECLIPSE FOUND. (Aug. 15, 2011), http://www.eclipse.org/org/documents/Eclipse%20BYLAWS%202011_08_15%20Final.pdf.

[27] *See* Spaeth et al., *supra* note 21, at 419.

[28] A complete list of Eclipse's member companies can be found at *Explore the Eclipse Membership*, ECLIPSE FOUND., http://www.eclipse.org/membership/exploreMembership.php (last visited Jan. 17, 2012). The list includes various entities (*e.g.*, publishing houses) that seldom write code. For a list of companies that have actually made code deposits, see *Eclipse Dashboard*, ECLIPSE FOUND., http://dash.eclipse.org/dash/commits/web-app/summary.cgi (last visited Jan. 17, 2012). More than two dozen nonmember companies have also made code deposits over the life of the project.

since 2007.[29] On the other hand, non-IBM companies have incorporated Eclipse into a host of new products.[30] This has made Eclipse the leading tool for developing software for mobile devices like cell phones, GPS units, and MP3 players.[31] Eclipse is also widely used to develop enterprise software for data-intensive problems like train scheduling and banking.[32]

Why do companies participate? The "business advantages," according to Eclipse, depend on cost sharing.[33] At first blush, this rationale may sound homely to readers accustomed to claims that OS produces software more efficiently than CS methods,[34] is better at finding and fixing bugs,[35] or generates streams of clever ideas from users.[36] But consider the math: OS advocates would be overjoyed if they could cut costs or increase quality by, say, 20 percent.[37] From a business standpoint, this means that companies could now pay eight workers instead of ten. At first glance, cost sharing, in which ten workers still do the work of ten, seems much less wonderful. But consider what happens when just two companies share. Now each partner pays just *five* workers. And these savings keep growing for each

---

[29] E-mail from Mike Milinkovich, Executive Director, Eclipse Foundation, Inc., to author (Jan. 4, 2012) (on file with author).

[30] Examples included Dev Rocket (Monta Vista), Luminosity (LynuxWorks), Neutrino (QNX Systems), Nucleus Edge (Accelerated Technology), Platform Express (Mentor Graphics), Tau (Telelogic Eclipse Foundation, Inc.), Struts Studio and JSF Studio (Exadel), MyEclipse (Genuitec), WebSphere Studio (IBM), C++ Compiler 8.1 for Linux (Intel), Kinzan Studio, NitroX (M7), Graphics Nucleus Edge (Mentor), SuSE SDK (Novell), Dev Suite (Palm), JTest (Parasoft), Designer (PureEdge), Momentics (QNX), Developer Suite (Red Hat), NetWeaver Studio (SAP), Xtensa Xplorer IDE (Tensilica), Time Storm IDE (TimeSys), and Workbench (Wind River). Robert Day, *Eclipse Heralds True Interoperability*, ELECTRONIC ENGINEERING TIMES (Apr. 3, 2006), http://www.eetimes.com/electronics-news/4059525/Eclipse-heralds-true-interoperability; Darryl K. Taft, *Eclipsing Challenges; Group's Directory Takes on Business, Technology*, ALLBUSINESS.COM (Nov. 15, 2004), http://www.allbusiness.com/technology/software-services-applications-open/13446070-1.html.

[31] Taft, *supra* note 30.

[32] *See Rich Client Platform Applications*, ECLIPSE FOUND., http://www.eclipse.org/community/rcp.php (last visited Jan. 17, 2012).

[33] More precisely, Eclipse says that it allows "individuals and companies to collaborate on projects that would be difficult to achieve on their own." *Eclipse Public License (EPL) Frequently Asked Questions*, ECLIPSE FOUND. http://www.eclipse.org/legal/eplfaq.php (last visited Jan. 17, 2012). Eclipse adds that its model has the further "technical advantage of turning users into potential co-developers." *Id.*

[34] RAYMOND, *supra* note 4, at 129–32.

[35] *Id.* at 33–36.

[36] *See* ERIC VON HIPPEL, DEMOCRATIZING INNOVATION 93–102 (2005).

[37] For a sense of the still-embryonic research on the comparative efficiency of OS and CS methods, see Stefan Koch, *Profiling an Open Source Project Ecology and Its Programmers*, 14 ELECTRONIC MARKETS 77 (2004), and Jai Asundi, *The Need for Effort Estimation Models for Open Source Software Projects*, *in* FIFTH WORKSHOP ON OPEN SOURCE SOFTWARE ENGINEERING 1, 1–3 (2005).

firm that joins the collaboration thereafter.[38] On any reasonable assumption, then, we expect sharing to dominate the business case for open source. What kinds of companies join Eclipse? Table 1 profiles the top ten companies that have donated software to the Project from 2001 to 2010.[39] These can be conceptually divided into four groups:

> *IBM.* Despite sharing, it remains true that IBM has contributed far more effort than any other company. This has included supplying one-fourth (24.3%) of all programmers and two-fifths (42.6%) of all software deposits between 2001 and 2010.[40] Even so, IBM's glass is more than half full: After all, nearly three-fifths of all deposits were paid for by someone else.
>
> *Other Dominant Companies.* The next four companies in Table 1—Oracle, Intalio, Cloudsmith and Actuate—contributed an additional 17 percent of all software deposits. Together with IBM, this shows that about 60 percent of the total Eclipse effort between 2001 and 2010 has come from five large players. To this point, the picture is not too different from a conventional joint venture.
>
> *Small Companies.* Eclipse's records show that at least eighty-six additional companies have contributed "commits" since 2001.[41] While the data are uncertain, these donations probably account for roughly 30 percent of all commits.[42] Thirty-four of these companies contributed at least 10,000 commits each.[43]

[38] Of course, these estimates assume that OS sharing is perfect. This is plainly untrue. Most notably, game theory suggests that OS collaborations should suffer from so-called "game-of-chicken" effects in which each participant delays creating software in hopes that somebody else will do the work for them. In equilibrium this reduces, but does not eliminate, OS sharing. *See, e.g.*, Justin Pappas Johnson, *Open Source Software: Private Provision of a Public Good*, 24 J. ECON. & MGMT. STRATEGY 637, 641–44 (2002).

[39] There are numerous ways to measure software donations. This article relies on the number of "commits," *i.e.*, separate instances in which a programmer deposits new or revised code in the project repository.

[40] Eclipse data from 2001 through 2010 provided by Mike Milinkovic and Wayne Beaton (Eclipse Foundation) and analyzed by Severin Weingarten (Friedrich Schiller University, Jena) [hereinafter Weingarten data]. Weingarten describes how this data was compiled in Severn Weingarten, On the Economics of Commercial Open Source Software: the Case of the Eclipse Project (Mar. 31, 2012) (unpublished Master's thesis, Fredrich-Schiller University Jena) (on file with Utah Law Review).

[41] Weingarten Data, *supra* note 40.

[42] The Eclipse data record commits by hundreds of individual programmers with no known employer. This figure is misleading, however, because many "[p]eople categorized under 'individual' may be associated with a company, just not an Eclipse member company that has signed a Member Committer Agreement." *Dash Project/Commits Explorer*, ECLIPSE FOUND., (Dec. 20, 2010), http://wiki.eclipse.org/Dash_Project/Commits_Explorer.

> *Traditional OS Participants.* The remaining Eclipse deposits come from sources that resemble traditional, noncommercial OS communities. These include nonprofits, academic organizations, and hundreds of individuals.[44] While these participants account for relatively few deposits—probably fewer than 10 percent—this may understate their value as lead users facing unusual computing problems. Similar user communities are known to be a valuable source of bug patches and new product ideas.[45]

Sharing is not the whole story, of course. Companies must also find a way to recover their investments. Table 1 profiles the top ten corporate software donors since 2001. Strikingly, all ten companies earn their living by making and selling CS products like hardware, software, or IT consulting services. Directly or indirectly, these sales pay for Eclipse.[46]

Making Sharing Work: Collaboration Architecture. The argument that Eclipse and other OS collaborations are driven by sharing deserves a closer look. After all, CS firms can also share costs by signing joint venture agreements. Why should OS work any better? One partial answer is that ordinary joint ventures are deeply flawed.[47] The main reason is that members usually find it difficult to monitor what their partners—or even their own employees—are doing. This leads to recurring principal-agent problems in which participants try to (a) hijack joint venture research in directions that favor themselves, or (b) shirk work altogether. On the other hand, OS methods have a well-deserved reputation for transparency. Could this cure the principal-agent problems that afflict CS sharing? IBM and its Eclipse partners are clearly betting that it will.

To estimate the size of this effect, consider that just 10 percent of the programmers listed as "individuals" contributed three-quarters (74%) of all "individual" deposits between 2001 and early 2010. *Eclipse Dashboard*, ECLIPSE FOUND., http://dash.eclipse.org/dash/commits/web-app/summary.cgi (last visited May 3, 2012). Furthermore, each of these contributors donated at least 10,000 commits apiece. Such effort strongly suggests a corporate sponsor. We therefore conclude, somewhat speculatively, that unpaid volunteers supplied fewer than 10 percent of all Eclipse commits between 2001 and 2010.

[43] Weingarten data, *supra* note 40.

[44] At least five academic and nonprofit institutes donated commits from 2001 to 2010. The largest were CEA LIST (1.0%) and INRIA (0.2%). Weingarten data, *supra* note 40.

[45] *See* VON HIPPEL, *supra* note 36, at 93–95.

[46] This pattern is similarly evident in the Linux collaboration as well as various OS projects (*e.g.* Android) that write software for use in mobile telephones. Readers can find a detailed statistical description in Barnett, *supra* note 8, at 1902–10, 1924.

[47] There is now a large literature on why conventional joint ventures fail. *See, e.g.*, SUZANNE E. MAJEWSKI & DEAN V. WILLIAMSON, INCOMPLETE CONTRACTING AND THE STRUCTURE OF COLLABORATIVE R&D AGREEMENTS (2003), *available at* http://www.law.harvard.edu/programs/olin_center/corporate_governance/papers/03.majwski-williamson.incomplete-contracting.pdf (examining how inability to completely specify R&D activities in advance leads to inefficient sharing).

As usual, the devil is in the details. The Eclipse Foundation tries to address transparency issues at two levels. The first set of solutions relates to high-level decision-making and would not be out of place in a traditional joint venture. Overall governance is committed to the Eclipse Foundation's Board of Directors which includes (a) representatives of fifteen large corporate sponsors, and (b) six elected members representing the committer and add-on communities.[48] The Board sets the Foundation's operating budget, decides which new software projects to develop, and can also amend the Foundation's OS license.[49] Because Board membership is diverse, no single interest group—including IBM and other large players—has nearly enough votes to hijack the collaboration.[50]

The Eclipse Foundation's second set of strategies for ensuring transparency takes a page from traditional OS methods. Beneath the Board level, companies play little or no formal role in prioritizing and performing work.[51] Instead, volunteers are encouraged to discard their corporate affiliations and interact as individuals. This is reinforced by governance rules that vest leadership in individual programmers ("committers") whose status has nothing to do with company affiliation and follows programmers when they change employers. Committers are said to be chosen on the basis of "meritocracy," "contributions," and "peer acclaim."[52]

Superficially, consigning production to a crowd of unsupervised programmers sounds like a formula for disaster. On the other hand, we have seen that traditional outside monitoring probably would not work in any case. This makes it sensible to rely on collaboration insiders who know the project first-hand. This strategy can only succeed, however, if insiders' incentives are aligned with their employers' goals. Commercial OS does this in several ways:

Efficiency Wage. Probably the most obvious strategy is for employers to pay insiders above-market wages and then fire them if the collaboration

---

[48] The current Board includes representatives from SAP, Talend, Actuate, OBEO, IBM, Innoopract, Oracle, CA Technologies, BREDEX, itemis, Sonatype, and six individuals elected by the committer and add-on provider communities. *Eclipse Foundation Board of Directors*, ECLIPSE FOUND., http://www.eclipse.org/org/foundation/directors.php (last visited Jan. 17, 2012). Professor Barnett has emphasized that many other OS collaborations including Linux, Ubuntu, GNOME, Firefox, and Apache similarly rely on independent boards to limit opportunism by individual sponsors. Barnett, *supra* note 8, at 1899–1900.

[49] The Board also hires and fires the Foundation's director, sets membership fees, terminates and reinstates members, and sets IP and antitrust policy. *See Membership Rights*, ECLIPSE FOUND., http://www.eclipse.org/membership/become_a_member/memberRights.php (last visited Jan. 17, 2012).

[50] As in a private corporation, fundamental changes to the bylaws and other ground rules must be approved by three additional classes of voting members along with programmers who have achieved leadership ("Committer") status within the project. *Id.*

[51] *Eclipse Development Process*, ECLIPSE FOUND., § 4.7, http://www.eclipse.org/projects/dev_process/development_process_2011.php.

[52] *Id.* § 4.1.

fails to deliver value. These "efficiency wages" create a strong incentive to please employers.[53] They also work indirectly by encouraging lower-level insiders to work hard so that they, too, can receive efficiency wages in the future.

Collaboration Assets. Insiders can usually extract rents from the collaboration over and above the value of their labor.[54] These assets include (a) the added productivity that collaboration volunteers gain from a long history of working together, (b) the ability to extract programming effort from individual volunteers pursuing traditional OS incentives like education or signaling,[55] and (c) money and in-kind contributions from corporate sponsors to the collaboration as a whole.[56] By definition, these assets and rents are only as durable as the collaboration itself. This gives insiders a powerful incentive to keep sponsors happy.

Self-Policing. Because OS collaborations are transparent to insiders, no single insider can shortchange a sponsor without the others noticing. But why should the other insiders permit this? After all, their income also depends on the collaboration's reputation for delivering value. This gives each insider a direct stake in seeing that all sponsors receive value.

Are any of these arguments correct? The jury is still out.[57] Nevertheless, companies have clearly found them sufficiently persuasive to justify ambitious

---

[53] For a description of efficiency wages, see, for example, WOLFGANG FRANZ & FRIEDHELM PFEIFFER, CTR. FOR EUROPEAN ECON. RESEARCH, THE RATIONALE FOR WAGE RIGIDITY: SURVEY EVIDENCE FROM GERMAN AND US FIRMS 3 (2003), *available at* http://ftp.zew.de/pub/zew-docs/dp/dp08097.pdf.

[54] Lerner & Tirole, *supra* note 5, at 217.

[55] Individual programmers often donate unpaid labor to commercial OS collaborations as a way of advertising their skills to prospective employers. *See, e.g.*, Lerner & Tirole, *supra* note 5, at 217. Here, the prospect of a future job acts as a prize to induce current effort.

[56] Typical examples include paying for workshops and conferences, purchasing hardware, and relicensing existing CS code under the GPL. Joel West & Siobhan O'Mahoney, *Contrasting Community Building in Sponsored and Community Founded Open Source Projects*, 38 HAW. INT'L CONF. ON SYS. SCI. 8 (2005), *available at* http://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1001&context=org_mgmt_pub.

[57] The steep disparity in effort between the Eclipse Foundation's first four corporate contributors and the remaining 178 companies suggests that OS sharing is highly imperfect. This may be because companies intentionally delay effort in hopes that someone else will do the work. *See, e.g.*, Johnson, *supra* note 38, at 641–44 (presenting model in which OS developers balance the expected costs and benefits of writing new programs against the chance to free-ride if someone else does). Alternatively, small companies may sometimes withhold effort knowing that their contributions will be miniscule in any case.

experiments like Eclipse. Until we learn differently, then, it is only prudent to set policy on the assumption that commercial OS sharing is here to stay. This makes it important to understand how OS affects the economy along with the challenges it poses for judges and policymakers.

III. THE OS/CS TRADEOFF: COMPETITION, SHARING, AND CARTELIZATION

The rise of commercial OS poses a complex investment problem for companies. On the one hand, OS sharing dramatically cuts each company's individual software development costs. On the other hand, shared code strengthens its competitors.[58] This can reduce profits to the point where OS no longer makes sense. Finally, each OS company offers consumers the same shared software as every other OS company. This leads to a de facto cartel that shelters companies from competition and limits their incentives to invest in software.

This Part explores OS economics in detail. It begins by asking how companies in a traditional all-CS industry decide how much to invest in software. The Article then explores how sharing changes this analysis for an all-OS industry before finally turning to mixed markets in which OS and CS companies compete with one another. It demonstrates that strong CS competition can force OS collaborations to produce far more software than a pure-OS industry would. However, this Part also shows that this condition seldom occurs in practice. Instead, market forces tend to produce information technology ("IT") ecosystems in which there are not enough CS firms to overcome the cartel effect. This systematically reduces the effort that OS firms would otherwise invest in shared code production. The resulting gap between OS software's potential and its actual performance poses a central challenge for judges and policymakers.

A. How CS Firms Invest

This Part begins by asking how companies in a traditional, all-CS industry decide how much to invest in software. Consider, for the sake of definiteness, a simple scenario in which companies create software in Year One, and then manufacture and sell a product that contains the software in Year Two.[59] This Year

---

In both cases, we should expect (a) large companies to shoulder more than their "fair share" of effort, and (b) total collaboration effort to be smaller than it otherwise would be.

[58] The Eclipse Foundation goes out of its way to warn prospective members that it "provides the same opportunity to all" and that "there are no rules to exclude any potential contributors which include, of course, direct competitors in the marketplace." *Eclipse Development Process*, *supra* note 51, § 2.1.

[59] For a more formal version of the analysis in this section, see Sebastian von Engelhardt & Stephen M. Maurer, *The New (Commercial) Open Source: Does It Really Improve Welfare?* 10–27 (Goldman School of Public Policy, Working Paper No. GSPP10-001, 2010), *available at* http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1542180. Additional technical details can be found in Sebastian von Engelhardt, *Quality Competition or Quality Cooperation?: License-Type and the Strategic Nature of Open Source vs.*

Two product can be a chattel (for example, a cell phone), a service (for example, technical support), or CS software.[60] For simplicity, this Article assumes that the quality of the Year Two product depends entirely on how much software it contains. Readers should note that this scenario includes two business decisions: (a) How much to invest in shared software development ("quality") in Year One, and (b) How many goods and services to manufacture and sell in Year Two. Each of these decisions provides a separate and distinct channel for companies to compete.

Note that a CS company will only invest if its expected profit from sales in Year Two is large enough to cover *both* the costs of making the product in Year Two *and* its software investment in Year One. Consider its Year Two profits first. These can be calculated using methods that are routinely taught in college economics courses.[61] The result is more or less what you would expect. As competition increases, each CS firm produces and sells more goods. Because prices fall, however, its total profits decline. We will refer to this dynamic as "Year Two competition" in what follows.[62]

The Year Two solution, in turn, sets up the Year One analysis. While Year Two profits clearly depend on the quantity of goods sold, they also depend on the quality of those goods, that is, how much software is written in Year One. Detailed calculation[63] shows that CS companies' Year Two profits (a) increase with the amount of software they write in Year One, and (b) decline when their rivals create competing software. We will refer to these effects as "Year One competition."

Closed Source Business Models 7–19 (Jena Economic Research Paper No. 2010-034, 2010), *available at* http://pubdb.wiwi.uni-jena.de/pdf/wp_2010_034.pdf. Doctors Llanes and de Elejalde use a similar analysis to investigate how firms choose between OS and CS business models. Gastón Llanes & Ramiro de Elejalde, *Industry Equilibrium With Open Source and Proprietary Firms*, 10–20 (Harvard Business School, Working Paper No. 09-149, 2009), *available at* http://papers.ssrn.com/abstract=1425549.

[60] Formally, the only limitation is that the product must be "excludable," *i.e.*, that sellers can control who uses it. Physical goods are clearly excludable since ordinary property law lets sellers control access. Similarly, service-providers can stop supplying users any time they choose. The case is subtler for software that consumers can copy at little or no cost. Despite this, companies can usually achieve reasonable excludability by using technical protections, invoking intellectual property law, and/or withholding source code and tech support.

[61] Briefly, each company calculates how many goods "*x*" to manufacture for every possible industry-wide output "*X*." The problem is then solved by finding the (at most) handful of cases in which the calculated *x*'s for every company add up to a self-consistent *X*. *See, e.g.*, WALTER NICHOLSON, MICROECONOMIC THEORY: BASIC PRINCIPLES AND EXTENSIONS 639–44 (6th ed. 1995).

[62] von Engelhardt & Maurer, *supra* note 59, at 14–24.

[63] *Id.*

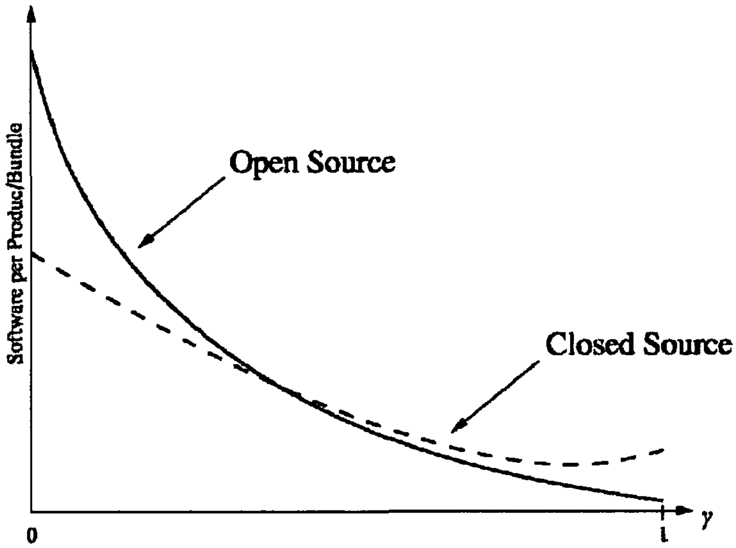

Figure 2: OS and CS Investment *vs.* Competition in Year Two[64]

Figure 2 summarizes these arguments. Here, the horizontal axis denotes how similar these companies' Year Two products are to one another with the degree of similarity increasing from left (completely different) to right (completely interchangeable). Readers should note, however, that product similarity also increases competition. For this reason, we will usually think of the horizontal axis as an index of Year Two competition.[65] Thus we can interpret the dashed line as showing how much software a typical CS company creates in Year One as a function of expected competition in Year Two. Production is highest on the left-hand side where companies expect to face little, if any, Year Two competition. In this case, the Year Two monopoly acts like an additional intellectual property incentive, giving the company maximal freedom to raise prices if and when consumers demand more of its software-enhanced products. Conversely, CS software production falls as competition increases. This is because the company knows that its ability to charge higher prices will be limited by the existence of similar, equally improved competing products

But this is not the whole story. Instead, CS software production levels off and even starts to rise again in right-hand portion of Figure 1. How is this possible? The answer is quality competition. When products are nearly interchangeable, even tiny quality differences can persuade consumers to switch from one company to another. This leads to a kind of arms race in which each company writes large amounts of software to keep its rivals from stealing customers. Like real arms races, these investments cancel each other out, leaving each company no better off—and considerably poorer—than it was before. This process is painfully

[64] Adapted from *id.* at 16.

[65] Competition also depends on the number of firms in the market. This effect is qualitatively similar to product similarity and we will normally ignore it in what follows.

familiar in Silicon Valley where CEOs frequently complain about razor-thin margins for "commoditized" (that is, generic) software.[66]

It is easy to understand why Silicon Valley CEOs should see "commoditization" as a disaster. Their business models center on being the first—and for a time, the only—company to deliver the next breakthrough product ("Killer App") to consumers.[67] These temporary monopolies are far more lucrative than creating new versions of existing products.[68] But what should the rest of us think? Outside Silicon Valley, most products are already commoditized —that is, they are produced by multiple competing companies. But in that case, our usual antitrust intuition suggests that commoditization is good for society. And indeed, generic economic models of CS production suggest that the parameters under which commoditization would hurt industry more than it helps consumers are extremely narrow.[69] While it is theoretically possible for commoditization to hurt society as a whole, such cases are probably rare and may not exist at all.[70]

B. How OS Firms Invest

Now consider how a company in an all-OS industry invests. Start by noticing that once the software exists, an OS company's Year Two choices are the same as a CS company's. This means that any differences must relate to the OS company's Year One strategy. Here, sharing introduces two important changes. On the one hand, it almost always reduces per-firm costs to below the CS case. On the other, OS sharing also dilutes the payoff from investing. This is because more software not only enhances the investing company's own products, but also makes its competitors' products more attractive.

To see how these effects interact, consider the solid line in Figure 1. OS investments are a no-brainer on the left hand side, where competition is weak. This

---

[66] *See, e.g.*, Ian Murdock, *Open Source and the Commoditization of Software*, *in* OPEN SOURCES 2.0, at 91 (Chris Dibona et al. eds., 2006), *available at* http://commons.oreilly.com/wiki/index.php/Open_Sources_2.0/Open_Source:_Competition_and_Evolution/Open_Source_and_the_Commoditization_of_Software. Commoditization appears to be a natural outcome for models based on Cournot competition. Interestingly, models based on Bertrand competition seem to lack this feature. (Professor Prasad Krishnamurthy, personal communication). This sort of disagreement between models is not unusual. The fact that Silicon Valley executives routinely complain about "commoditization" suggests that Cournot models are on the right track.

[67] *See* Murdock, *supra* note 66, at 93.

[68] They may even be socially justified if the lure of large rewards induces companies to develop new innovations earlier or more reliably. *See, e.g.*, SUZANNE SCOTCHMER, INNOVATION AND INCENTIVES 112 (2004).

[69] von Engelhardt & Maurer, *supra* note 59, at 22–24.

[70] The reason is that companies in concentrated industries can usually take steps to mitigate commoditization. Here, the normal strategy is for each company to optimize its product for a different submarket. This suggests that the welfare losses associated with extreme commoditization will only occur in those rare industries where technical or market considerations force companies to make nearly identical products.

is because companies can split costs without threatening each other's revenues. What could be better?[71] On the other hand, revenues decline as products become more interchangeable and quantity competition increases. This explains why OS companies, like their CS counterparts, produce less and less software as competition increases along the right-hand side of Figure 1.[72]

Even so, there is a surprise. Unlike the CS case, OS output never encounters a commoditization crisis in which an arms race between firms revives incentives to produce software. Instead, OS software production never stops falling. How can this be? The reason is sharing. By definition, no company in an all-OS industry can offer consumers better shared software than any other company.[73] This suppresses quality competition much as a formal cartel would. (In fact, the incentives to invest are actually worse than a cartel.[74]) While OS companies invest in software, then, they stop writing sooner than they would if quality competition existed.[75] This leads to the usual monopoly result—an undersupplied product.[76]

---

[71] Actually, one *can* imagine something better. As Dr. Schmidke points out, many goods are complements so that improvements to one product increase demand for both. In this situation, firms that invest in R&D confer benefits not only on themselves but also others. Since only the former are compensated, this leads to underinvestment. OS sharing gets around this externality by letting companies subsidize each others' products. Richard Schmidtke, *Private Provision of a Complementary Public Good* 8–9 (CESifo, Working Paper No. 1756, 2006).

[72] von Engelhardt & Maurer, *supra* note 59, at 8.

[73] This statement assumes that companies cannot write separate, proprietary modules that "turbocharge" the shared codebase's performance. This is reasonable. First, OS licenses typically contain "viral" terms that limit how closely proprietary modules can interact with the underlying codebase. Second, letting companies divert improvements into proprietary modules would immediately cripple OS sharing. Indeed, sharing stops entirely in our model. Knowing this, we would expect consciously parallel conduct to avoid the practice. Finally, the idea of using separate proprietary modules to turbocharge performance is often impractical on marketing and technical grounds. We return to these issues *infra* Part V.

[74] Ironically, OS collaborations actually produce *less* software than an explicit cartel would. The reason is that a well-designed cartel maximizes industry-wide profits. By contrast, participants in an OS collaboration only care about their own profits and ignore their work's impact on the profits of other members. This reduces their incentive to produce code in the first place. Maurer & Engelhardt, *supra* note 59, at 18, 39–40.

Readers should note that Dr. Llanes and his co-authors do not find cartel effects in their models of OS/CS investment. Llanes & de Elejalde, *supra* note 59, at 34–35; Ramon Casadesus-Masanell & Gaston Llanes, *Mixed Source* 5–6 (NET Institute, Working Paper No. 09-06, 2009). This seems to be an artifact of the authors' assumption that companies make their Year One and Year Two decisions in one single instant. Importing this assumption into the Maurer and von Engelhardt model likewise suppresses cartel effects. While mathematically convenient, the Llanes *et al.* decision fails to capture the very different lead times for writing software compared to making physical goods.

[75] More specifically, companies stop investing when their expected profit is largest. Here, the basic intuition is that (a) consumer willingness to pay (and hence revenue) is approximately linear in program size, but (b) development costs increase steeply for large

We have come to a paradox. On the one hand, OS companies' ability to share costs means that they can produce far more software than any CS competitor. On the other, this same sharing creates a de facto quality cartel that suppresses OS companies' incentives to invest in the first place. We therefore expect that OS collaborations facing stiff Year Two competition will invest less effort in writing software than a CS company would. This will still be good for society in the usual case where the shared, collaboration-wide OS effort produces more software than any CS company can fund on its own. At the same time, it remains true that OS companies would have invested still more absent the cartel effect. For OS-enthusiasts and policymakers, the glass is only half-full.

C. Can Competition Fix the Problem?

We have seen that OS promotes sharing but suppresses quality competition among collaboration members. On the other hand, antitrust law reflects the belief that competition cures most ills.[77] Can CS companies deliver the missing quality competition? Simple calculations are encouraging. For example, generic models suggest that IT industries deliver maximum benefits when 15–20 percent of companies practice OS and the rest are CS.[78] [Figure 3] Reassuringly, this

---

software projects. This suggests that OS companies will normally stop writing software once development costs start to increase faster than expected revenues. CS companies do not have this option. Instead, quality competition forces them to continue investing in new software despite "commoditization."

[76] In retrospect, this should not surprise us. Scholars have worried since the 1980s that conventional CS sharing (*i.e.* joint R&D ventures) could suppress competition in innovation. This literature is often strikingly similar to the current analysis. For example, Professor Katz argued in 1986 that R&D joint ventures eliminated duplication but also reduced members' incentives to invest. Michael L. Katz, *An Analysis of Cooperative Research and Development*, 17 RAND J. ECON. 527, 527–543 (1986). The latter effect could, however, be mitigated where (a) joint venture members' downstream products faced minimal competition, or (b) the joint venture was just one of several R&D programs operating in the industry. *Id.* Katz later served as Deputy Assistant Attorney General for Economic Analysis from 2001 to 2003. Some courts have similarly expressed concern that joint ventures could operate to suppress R&D incentives. *See, e.g.*, Addamax Corp. v. Open Software Found., Inc., 888 F. Supp. 274, 282–86 (1995) (joint R&D venture could potentially cartelize software production by participants), *aff'd*, 152 F.3d 48, 52 (1st Cir. 1998).

[77] Darren Bush, *Mission Creep: Antitrust Exemptions and Immunities As Applied to Deregulated Industries*, 2006 UTAH L. REV. 761, 806.

[78] von Engelhardt & Maurer, note 59. Careful readers will note that OS output in Figure 3 actually peaks when OS companies constitute about 10 percent of the market. Social benefits would also peak at this point if they depended solely on the amount of OS software produced. In fact, however, CS software also generates benefits by serving consumers who prefer to pay low prices for low quality software. This explains why social benefits for the combined (OS + CS) market only peak when 15–20 percent of all companies pursue OS business plans. *Id.* at 27.

percentage is more or less identical to the rule of thumb that the Department of Justice usually invokes when measuring cartelization threats, for example, in mergers.[79]

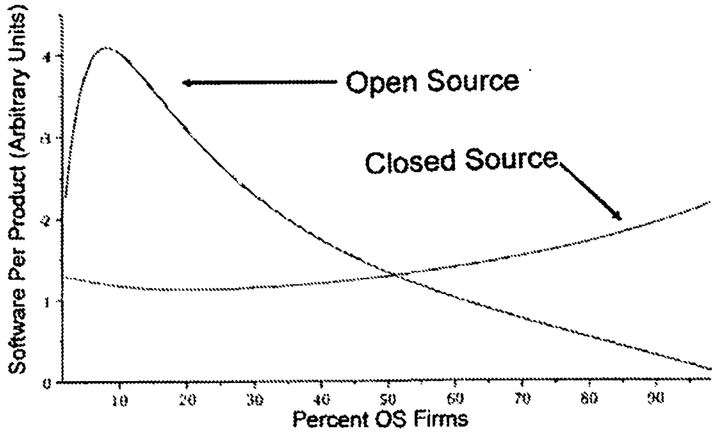

Figure 3: Software Production *vs.* Percent of Industry Members Adopting OS Methods.[80]

Finally, Figure 3 assumes a scenario in which a single OS collaboration competes with multiple CS companies. However, one can imagine still more favorable scenarios in which several OS collaborations compete with one another. Such situations promise the same level of competition as Figure 3 with radically increased sharing. For example, the Eclipse collaboration currently competes with both proprietary (JDeveloper) and open developer tools (NetBeans, IntelliJ).[81] Similar examples in which open products compete against each other include Apache Harmony versus OpenJDK (Java runtimes),[82] Pentaho versus Jaspersoft (business reporting software),[83] and Jetty versus Tomcat (web servers).[84] The

[79] Merger review is triggered at a Herfindahl index of 1800. U.S. DEP'T OF JUSTICE & THE FED. TRADE COMM'N, HORIZONTAL MERGER GUIDELINES 15–16 (rev. 1997). This overlaps the 15–20 percent window calculated by von Engelhardt & Maurer, *supra* note 59, at 27.

[80] von Engelhardt & Maurer, *supra* note 59, at 21.

[81] *Oracle JDeveloper*, ORACLE, http://www.oracle.com/technetwork/developer-tools/jdev/overview/index.html (last visited Jan. 17, 2012); NETBEANS, http://netbeans.org/ (last visited Jan. 17, 2012); INTELLIJ IDEA, http://www.jetbrains.com/idea/ (last visited Jan. 17, 2012).

[82] APACHE HARMONY, http://harmony.apache.org/ (last visited Jan. 17, 2012); OPENJDK, http://openjdk.java.net/ (last visited Jan. 12, 2012).

[83] PENTAHO, http://www.pentaho.com/ (last visited Jan. 12, 2012); JASPERSOFT, http://www.jaspersoft.com/regForms/download-30day-trial/index.php?leadsource=PPC-Google&gclid=CPTIz8q1160CFQhjhwodsmnroQ (last visited Jan. 12, 2012).

picture for Content Management Systems—in which OS collaborations Plone, Drupal, PostNuke, and Joomla all compete with one another—is even more encouraging.[85] The main drawback is that some or all of these projects could fall by the wayside over time.

D. Market Imperfection No. 1: Lock-In

We have seen that the benefits of sharing are largest in markets where CS companies outnumber their OS competitors by roughly four to one.[86] Do market forces guarantee this ratio? The answer is almost certainly "no." One reason is that infant industries will often be born in an All-OS or All-CS state.[87] Such industries can become "locked in" so that mixed OS/CS markets never emerge.[88] To see why, consider an infant industry that consists entirely of CS companies. Clearly, OS companies will only enter the market if they expect to earn a profit. However, detailed calculation shows that this may not be possible in markets where Year Two competition is strong.[89] For this reason, we expect many All-CS markets to exclude OS indefinitely.[90] In a few cases, incumbents may deliberately select CS so that would-be entrants cannot take advantage of low-cost OS codebases to enter the market. Here, we expect companies to choose CS despite the fact that OS sharing would yield higher profits in the short run.[91]

Lock-in can also occur in All-OS markets where Year Two competition is too weak for would-be CS entrants to earn a profit. Here too, we expect market forces to block the emergence of more balanced markets in which OS and CS companies compete with one another.

E. Market Imperfection No. 2: Too Many OS Firms

The models that predict lock-in require fairly special parameters.[92] This implies that mixed OS/CS markets should eventually emerge in many, if not most, cases. Even when this happens, however, the market will encourage too many

---

[84] JETTY, http://jetty.codehaus.org/jetty/ (last visited Jan. 12, 2012); APACHE TOMCAT, http://tomcat.apache.org/ (last visited Jan. 12, 2012).

[85] I am indebted to the Eclipse Foundation's Mike Milinkovich for these examples.

[86] *See supra* text accompanying notes 76–78.

[87] This result is more or less inevitable where the young industry consists of just one or two companies. Because OS sharing needs at least two companies, we naively expect more All-CS industries than All-OS ones.

[88] von Engelhardt & Maurer, *supra* note 59, at 28, 32–36.

[89] *Id.* at 15, 32–36. The reason for this result is apparent from Figure 2, which shows that OS collaboration investments go to zero as quantity competition increases.

[90] *Id.* at 14–16, 32–37.

[91] *Id.* at 19–22. The logic is reminiscent of predatory pricing, in which firms deliberately incur short-term losses in order to charge higher prices once their rivals have been driven from the market.

[92] *Id.* at 33–36.

firms to practice OS. The reason, once again, is the cartel effect, which suggests that any given group of OS companies will normally be more profitable than the same number of CS companies.[93] This implies that most companies will adopt OS business models. Figure 4 (dotted line) shows what happens in the case where entry continues until the profits of both OS and CS companies fall to zero. For most values of competition, the number of OS companies hovers at roughly 60–70 percent. However, these companies tend to be small so that OS's total market share remains low.[94] Conversely, the CS sector is dominated by a handful of large companies that serve most of the market. This pattern is frequently seen in real software markets.[95]

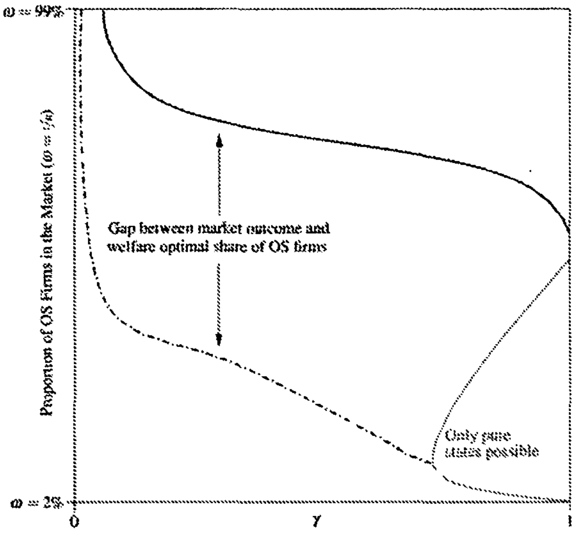

Figure 4: Ideal (Dashed) and Predicted (Solid) Percentage of Companies Practicing OS Business Models as a Function of Competition.[96]

This systematic over-supply of small OS firms has profound implications. Ideally, policymakers would like markets to occupy the left-hand side of Figure 3 where software output is greatest. Because of the cartel effect, however, we expect real OS collaborations to fall on the right-hand side and produce less software than

[93] *See supra* text accompanying notes 69–74.

[94] OS companies' small average size follows from the cartel effect. On the one hand, high profits increase the number of companies that practice OS. On the other hand, the cartel effect suppresses output. This means that average per-firm output must be small.

[95] The point is nicely illustrated by the hundreds of small companies that participate in commercial OS collaborations like Eclipse and PLONE. *See, e.g., Explore the Eclipse Membership*, ECLIPSE FOUND., http://www.eclipse.org/membership/exploreMembership.php (last visited Jan. 17, 2012); *Plone Service Providers*, PLONE.ORG, http://plone.net/providers/ (last visited Jan. 17, 2012).

[96] von Engelhardt & Maurer, *supra* note 59, at 31.

their CS competitors.[97] This anemic result is still better than having each OS company write wastefully duplicative software.[98] Nevertheless, the situation is far from ideal. Parts IV through VI will evaluate judges' and policymakers' options for closing this gap.

F. Beyond Simple Models.

So far, we have analyzed OS output and the cartel effect using deliberately generic assumptions.[99] Nevertheless, details matter. We close this section by asking how more complicated—but still plausible—assumptions could change our understanding:

1. Inefficient Sharing

We have assumed that OS sharing is efficient.[100] Though plausible, this remains to be seen. Less efficient sharing would reduce the benefits of OS while leaving the cartel effect unchanged. This suggests that judges and policymakers should scrutinize OS more closely where sharing is imperfect.

2. Idiosyncratic Consumers

We have assumed that products compete on both quantity and quality.[101] However, software consumers sometimes display loyalty to particular products despite large differences to price and/or quality.[102] Here, quality competition will likely be weak in any case.[103] In these circumstances, policymakers and judges

[97] Experience with real world markets confirms this intuition. Despite considerable argument, there is no clear evidence that OS code performs substantially better than equivalent CS products. The relevant literature is discussed at length in, for example, Maurer & Scotchmer, *supra* note 1, at 285–322.

[98] Economists usually refer to the wasteful duplication associated with conventional CS production as "business stealing" or "me-too products." *See, e.g.*, Joachim Henkel & Eric von Hippel, *Welfare Implications of User Innovation*, 30 J. TECH. TRANSFER 73, 78–79, 83 (2005).

[99] *See supra* text accompanying notes 56–94.

[100] *See supra* text accompanying notes 55–56.

[101] *See supra* text accompanying notes 56–94. Formally, the assumption leads to models that feature classically linear demand curves. Maurer and von Engeldhardt, *supra* note 59, at 10.

[102] Brand loyalty is commonly represented by rank-ordering products along an imaginary line according to their attributes. These models are usually constructed so that consumers who prefer a given set of attributes will often select "nearby" products over higher-quality, but more distant alternatives. For an application of these ideas to commercial open source, see Llanes & de Elejalde, *supra* note 59.

[103] Detailed calculations by Doctors Llanes and De Elejalde confirm this intuition for a model containing two technology sectors. *Id.* at 10–20.

could reasonably conclude that any additional damage from cartel effects is not worth worrying about.

3. *Multiple Technologies*

This Part has assumed for simplicity that product quality is derived from a single, indivisible technology called "software." In fact, quality usually depends on multiple technologies. This suggests that companies may choose to develop some technologies using OS sharing and others using CS methods. If so, the benefits of OS sharing—but also the drawbacks associated with OS cartels—will be diluted so that the basic OS/CS choice becomes less important.[104]

4. *Product Differentiation*

We have assumed that OS sharing reduces the wasteful duplication inherent in having each company write its own software.[105] However, duplication also increases the odds that at least one program will arrive sooner—or perform better—than the others. Moreover, duplicative programs may not be wasteful if each is tailored to a different user group. Judges and policymakers should be careful to look for these effects before they accept the benefits of OS sharing at face value.

5. *Limited Entry*

We have made the conventional assumption that new companies can and will enter the market until incumbents' profits fall to zero.[106] If entry is slow, however, it may make more sense to assume that the number of companies is fixed and ask how many will choose OS business plans. Fortunately, this new assumption leads to qualitatively similar results. We still expect the cartel effect to make OS

[104] This is indeed what happens in Doctors Llanes and De Elejalde's analysis when they split their technology sector into two pieces, only one of which is suitable for OS sharing. *Id.* The case may be different where technologies splinter into fragments that are too small for efficient R&D sharing so that each company takes responsibility for whichever project(s) offer the most value to its business. As Professor Henkel has emphasized, this boosts total investment and makes OS sharing more attractive. Joachim Henkel, *The Jukebox Mode of Innovation — a Model of Commercial Open Source Development* 10–11 (Danish Research Unit for Industrial Dynamics, DRUID Working Paper No. 06-25, 2006), *available at* http://papers.ssrn.com/sol3/papers.cfm?abstract_id=578142. Judges and policymakers should be especially tolerant of cartel effects in these circumstances.

[105] *See supra* text accompanying notes 55–74.

[106] *See supra* text accompanying notes 55–95.

companies systematically more profitable and therefore more numerous than CS companies.[107]

Based on this discussion, it is reasonable to think that more complex fact patterns will sometimes dilute or qualify our arguments. The basic themes, however, are robust. These include the benefits of OS sharing, the OS cartel effect, and the resulting oversupply of OS companies. The next three sections explore how judges and policymakers should exploit these insights.

IV. THE SHERMAN ACT (A): RULE OF REASON

Section 1 of the Sherman Act reaches "[e]very contract, combination . . . or conspiracy, in restraint of trade . . . ."[108] Under the rule of reason, judges and policymakers must intervene whenever an agreement's "anticompetitive effects on trade outweigh its procompetitive effects."[109] We have seen that OS cost sharing eliminates wasteful duplication and lets companies write far more software than they could before.[110] However, we have also seen that cartel effects will often suppress the second benefit.[111] Does this diminished OS still offer enough benefits to satisfy the rule of reason?

This Part reviews the antitrust rules for CS cost sharing (that is, joint ventures) and uses this baseline to develop a rule of reason analysis for OS collaborations. We will see that it is almost always possible to design OS collaborations that satisfy the rule of reason for operating systems. However, the rule of reason case for applications programs is much closer and should normally be viewed with suspicion. I also suggest two "safe harbor" exceptions in which OS collaborations should be presumed valid. The Part concludes with a short discussion of monopolization issues under Section 2 of the Sherman Act.[112]

[107] Examples of such models can be found in Llanes & DeElejalde, *supra* note 59, at 18, and Casadesus-Masanell & Llanes, *supra* note 74, at 19–25.

[108] 15 U.S.C. § 1 (2006). Liability does not depend on whether the challenged contract is memorialized in a single, overarching agreement or—as with OS licenses—a series of bilateral agreements. Interstate Circuit, Inc. v. United States, 306 U.S. 208, 208, 226 (1939) (inferring conspiracy from multiple bilateral agreements where parties agreed "knowing that concerted action was contemplated and invited").

[109] Nat'l Collegiate Athletic Ass'n v. Bd. of Regents of Univ. of Oklahoma, 468 U.S. 85, 104, 113 (1984) (agreements that raise prices or restrict output bear a "heavy burden" of showing that "apparent deviation" from normal free market principles is "competitively justifie[d]").

[110] *See supra* text accompanying notes 55–95.

[111] *See supra* text accompanying notes 71–75.

[112] 15 U.S.C. § 2.

A. Existing Joint Venture Law

There is still surprisingly little case law and commentary analyzing OS collaborations under the Sherman Act.[113] Fortunately, the antitrust treatment of CS sharing (that is, joint research and development ("R&D") ventures) has been extensively discussed, though seldom litigated.[114] For most of the twentieth century, antitrust authorities evaluated joint R&D ventures like any other cartel.[115] Debates over US competitiveness in the 1980s, however, led the US government, Congress, and scholars to a new appreciation that cost sharing could encourage companies to set more ambitious R&D goals. In the words of Professor Edward Correia:

> "[A] single firm is frequently unwilling to make the investment necessary to enter a market or solve a technological problem on its own, but it will invest a smaller amount in a collective effort with a better chance of success. If courts incorrectly assume that individual collaborators will engage in the same effort on their own the result is to discourage collaborative investment on the mistaken theory that there will be even more investment if firms act independently."[116]

This new appreciation for sharing was nevertheless tempered by fears that cartelization might limit companies' incentive to invest in R&D.[117] Even if sharing did suppress R&D, however, joint ventures would still satisfy the rule of reason unless these "disincentives" were "sufficiently strong" so that "the decreased competitive pressure to innovate will outweigh whatever economies of scale or other efficiencies are likely to be generated by the collaboration."[118] Tests of this criterion included asking (a) whether the agreement "eliminates innovations that would have been cost-justified in a competitive innovation market,"[119] (b) whether

[113] *But see* Wallace v. IBM, 467 F.3d 1104, 1107 (7th Cir. 2006) (holding that GPL provision setting license fees equal to zero was not predatory pricing because GPL "keeps price low forever" and cannot "lead to monopoly prices in the future").

[114] The federal government challenged just one joint venture on Sherman Act grounds during the entire twentieth century. Robert Pitofsky, *Antitrust and Intellectual Property: Unresolved Issues at the Heart of the New Economy*, 16 BERKELEY TECH. L.J. 535, 544–45 (2001) (citing United States v. Auto. Mfrs. Ass'n, 307 F.Supp. 617, 621 (C.D. Cal. 1969)).

[115] *See id.*

[116] Edward Correira, *Joint Ventures: Issues in Enforcement Policy*, 66 ANTITRUST L.J. 737, 760 (1998).

[117] *Id.* at 759.

[118] *Id.*

[119] 13 HERBERT HOVENKAMP, ANTITRUST LAW: AN ANALYSIS OF ANTITRUST PRINCIPLES AND THEIR APPLICATIONS § 2115, at 114 (2d ed. 2005); *see also* Northrop Corp. v. McDonnell-Douglas Corp., 705 F.2d 1030, 1052–53 (9th Cir. 1983) (holding that

the parties would "have developed [the product] more quickly alone,"[120] and (c) "what the parents would have done but for the joint venture."[121] More concretely, Congress and the Department of Justice also created a new safe harbor for joint R&D ventures.[122] This "five effort rule" created a presumption that joint R&D ventures are justified so long as they face competition from at least four other comparable programs. Strikingly, this new rule did not create an especially favorable test for CS sharing. Instead, it merely repeated the Department of Justice's usual rule of thumb for evaluating market concentration in merger cases.[123]

Formally, none of this marked a significant departure from previous learning. Nevertheless, the change was real. Prior to the 1980s, antitrust authorities had viewed CS sharing with suspicion.[124] This underlying attitude was now reversed by a new intuition that "[t]he optimal amount of innovation can ordinarily be achieved through competition or cooperative innovation ventures."[125]

B. *A Rule of Reason Approach to OS Collaborations*

We have seen that OS's chief benefit is cost sharing and that its main vice is cartelization.[126] Furthermore, these effects cannot be disentangled. Firms might not invest without the promise of shared code, but this same promise also ensures cartelization. Judges applying the rule of reason must therefore decide whether the (admittedly truncated) benefits of OS sharing justify the cartel effect. Part III has argued that this balance should normally favor OS. This suggests a kind of informal intuition—as in the joint venture case—that most OS collaborations can be designed in ways that satisfy the rule of reason. On the other hand, this judgment may not hold for more complicated fact patterns. Judges should be particularly alert to instances where the value of OS sharing is doubtful, for example where (a) OS sharing is inefficient, (b) otherwise duplicative CS

a joint venture to invent military aircraft that neither partner could develop separately was not a per se violation).

[120] *See* HOVENKAMP, *supra* note 119, at 118.

[121] Correia, *supra* note 116, at 760.

[122] National Cooperative Research and Production Act, 15 U.S.C. §§ 4301–4305 (2006); U.S. DEP'T OF JUSTICE, ANTITRUST ENFORCEMENT GUIDELINES FOR INTERNATIONAL OPERATIONS § 2.5 (1995), *available at* http://www.justice.gov/atr/public/guidelines/internat.htm.

[123] Professor Correia has pointed out that an industry comprised of five identically sized competitors has a Herfindahl index of 2000, whereas the Justice Deptartment scrutiny is triggered above 1800. Correia, *supra* note 116, at 759 n.84.

[124] *Id.* at 756–58.

[125] HOVENKAMP, *supra* note 119. This expansive proposition is said to be founded on economics research suggesting that even "a monopolist will not underinvest in research" in most real world situations. Correia, *supra* note 116, at 759 & n.82.

[126] *See supra* Part III.

programs have been optimized to serve different user groups, or (c) competing R&D programs provide useful redundancy against failure.

The potential damage from cartelization will also vary from case to case. OS collaborations are especially likely to satisfy the rule of reason where cartelization effects either do not matter or matter very little. At least three situations fit this description:

1. Limited Absorptive Capacity

All projects eventually encounter diminishing returns beyond which further investment offers little improvement. The question is whether cartelization chokes off investment before or after this point is reached. In general, courts should be more willing to tolerate cartelization where the evidence suggests the project cannot usefully absorb additional resources.

2. Low-Priority Projects

We have assumed that companies fund every project that offers a positive return on investment. In Silicon Valley, however, capital is almost always scarce so that only the most promising projects are funded. Cartelization only matters if it affects these favored projects; the rest will be discarded in any case.

3. Pathological Competition

We noted in Part III that it is possible to construct scenarios in which quality competition drives software production so far into diminishing returns that manufacturers lose more value than consumers gain. That said, such situations are almost certainly rare and courts should greet such claims with skepticism.

Deciding when pathological competition exists is clearly fact dependent. Nevertheless, the most important variable—the type of software under development—can be anticipated. For convenience, we consider three types of products: (a) operating systems, (b) breakthrough programs that establish entirely new categories of software ("Killer Apps"), and (c) new applications that perform mostly familiar functions ("Commoditized Apps").

a) Operating Systems

Operating systems provide the clearest candidate for a software product in which the downsides of cartelization can be safely ignored. Courts could sensibly conclude that existing operating systems are already deep into diminishing returns. First, their effect is mediated through application programs. This means that further improvements would exert, at best, only an indirect influence on the look, feel, and functionalities consumers actually receive. Second, operating systems often grow by absorbing existing functions from apps. The benefits of accelerating this process are probably small. Finally, the division of labor between apps and operating systems is technically flexible. For this reason, failures to develop operating system features can often be remedied at the apps level.

Engineers say that "operating systems should be open and applications programs should be closed."[127] Apart from the empirical observation that most OS collaborations develop operating systems, however, the policy reasons for this intuition are unclear. Our analysis validates the first ("operating systems") part of this statement. We now turn to the second part, that is, whether "application" programs should normally be closed.

b) Killer Apps

Part III's arguments were predicated on a tacit assumption that program quality depends almost entirely on the amount of code that is written. However, not all software derives its value from effort that "is sold by the pound."[128] Instead, application programs often derive significant earning power from clever new features.[129] Here, the most extreme case consists of so-called "Killer App" products that discover previously overlooked needs or invent new functionalities.[130] Any attempt to organize Killer Apps as an OS collaboration would therefore gain little from cost sharing while still suppressing incentives to innovate. This would almost certainly violate the rule of reason.

Fortunately, the threat is remote. Indeed, the empirical evidence suggests that OS collaborations avoid idea-driven products.[131] Part of the reason is practical—because innovative ideas are (by definition) unanticipated, it is almost impossible to write an OS agreement that defines which ideas must be shared. This destabilizes OS collaborations by encouraging members to develop their best ideas as CS products. There is also a deeper problem. Even if an OS cartel were possible, most Silicon Valley CEOs, venture capitalists, and shareholders probably prefer high-risk, large-payoff races to develop the next Killer App.[132] The prospect of small-but-predictable rewards does not appeal to them.[133]

---

[127] I am indebted to the late Professor Richard Newton for this observation.

[128] I am grateful to the Eclipse collaboration's Mike Milinkovich for coining this phrase.

[129] *See, e.g.*, CARL SHAPIRO & HAL R. VARIAN, INFORMATION RULES: A STRATEGIC GUIDE TO THE NETWORK ECONOMY 216 (Harvard Bus. Sch. Press 1999).

[130] *Id.*

[131] Krzyszetof Klincewicz, *Innovativeness of Open Source Software Projects*, NORWEGIAN U. SCI. & TECH., 23–26 (Aug, 10 2005), http://www.idi.ntnu.no/grupper/su/bibliography/pdf/newoyvh/klincewicz2005.pdf.

[132] *See* THE SILICON VALLEY EDGE: A HABITAT FOR INNOVATION AND ENTREPRENEURSHIP 96–111 (Chong-Moon Lee et al. eds, 2000).

[133] *See id.* Hardware and service providers are even less likely to want this trade. "Killer Apps" have historically been instrumental in persuading consumers to upgrade their computer systems. *See* Scannell, *supra* note 129.

c) Commoditized Apps

Over time, even the most unique products are imitated and become generic. At this point, the payoffs from making still more clever improvements become smaller and smaller. Companies may find it tempting to cartelize such software and refocus their cleverness on Killer Apps.

For now, it is hard to think of a convincing example in which OS has successfully cartelized an applications market. However, the potential is there. Consider, for example, the market for business desktops in which Microsoft's dominant CS desktop products currently face four OS competitors.[134] On the one hand, this five-way competition surely minimizes cartelization risk for the immediate future. On the other hand, it is easy to see how many of today's OS collaborations could merge or drop by the wayside. If this happens, the cartel effect would grow steadily stronger.

1) Proposed Safe Harbors

Our argument that society receives maximal value when roughly 15–20 percent of all software companies practice OS sharing[135] closely resembles the US Department of Justice's five effort rule for joint ventures. This is no coincidence. Part III has argued that the cartel effect provides the main brake on OS output. It only makes sense that the effect should be smallest in industries that satisfy the Justice Department's normal rule-of-thumb for judging market concentration.

The five effort rule will, however, need to be revised before it can be applied to OS collaborations. The reason is that our arguments do not depend on effort *per se*, but only on the number of OS companies which have joined the collaboration.[136] For this reason, an OS safe harbor should be based on membership rather than effort. This revised safe harbor would immunize any OS collaboration that included less than 20 percent of all current industry members.

It also makes sense to create a second safe harbor for OS collaborations that develop operating systems. Such collaborations usually derive large benefits from sharing while presenting only minimal cartelization risks.[137] Here, the main practical difficulty is the blurred line between operating systems and application programs. This would encourage OS collaborations to relabel application projects as "operating systems" in order to evade judicial scrutiny. Fortunately, this danger can be managed by limiting the safe harbor to programs that (a) are typically found

[134] The competitors are Sun's Star Office, Oracle Open Office, Gnome, and the K Desktop Environment.

[135] *See supra* Figure 4.

[136] Similarly, we have argued in Part III that the amount of effort that OS companies invest can vary dramatically depending on how much competition they face.

[137] *See supra* Part III.

in today's operating systems or (b) have become so generic that their inclusion in an operating system is no longer surprising.[138]

2) Section 2 Claims?

So far we have concentrated on the idea that OS might violate Section 1 of the Sherman Act.[139] This is reasonable since a Section 2 claim would require proof that violators already possessed a substantial share of the market.[140] OS companies hardly ever meet this test.[141]

Section 2 issues could still arise on the CS side. Here, the most likely issue is lock-in. We have seen that incumbents can sometimes deliberately adopt CS to block entry by new competitors.[142] Still, this behavior is unlikely to give rise to a Sherman Act violation. The reason lies in the well-settled case law holding that Section 2 does not create a duty to aid competitors.[143] In effect, plaintiffs would have to argue that the OS collaboration was an "essential facility," with the added twist that incumbents could not refuse to build such a facility in the first place. Given the disfavored status of essential facility claims under modern antitrust law,[144] this theory would almost certainly fail.

V. THE SHERMAN ACT (B): VIRAL LICENSES

As discussed above—the cartel effect notwithstanding—OS sharing is often justified under the rule of reason.[145] At this point, judges must accept some suppression of output. Further attempts to mitigate the cartel effect, if they come at all, will require government intervention.[146]

On the other hand, there is still much for antitrust courts to do. In particular, judges will need to make sure that whatever cartelization does occur is truly unavoidable. Here, the most urgent issues involve so-called "viral" licenses[147] that

[138] The second inquiry would not be free from controversy. Courts have been reluctant to second-guess defendants' claims that technical efficiency justifies bundling traditionally separate applications into a single program. *See* United States v. Microsoft Corp., 253 F.3d 34, 84 (D.C. Cir. 2001).

[139] *See supra* text accompanying note 28.

[140] *See* Spectrum Sports, Inc. v. McQuillan, 506 U.S. 447, 459 (1993) (requiring proof of a "dangerous probability that a [defendant] would monopolize a particular market").

[141] *See supra* Part III.D–E.

[142] *See supra* Part III.D.

[143] Verizon Commc'ns, Inc. v. Law Offices of Curtis V. Trinko, LLP, 540 U.S. 398, 410–11 (2004).

[144] For a recent review, see Robert Pitofsky et al., *The Essential Facilities Doctrine Under U.S. Antitrust Law*, 70 ANTITRUST L.J. 443, 443–45 (2002).

[145] *See supra* Part III.

[146] *See infra* Part VI.

[147] "Viral" or "copyleft" licenses contain terms that require "anyone who redistributes the software, with or without changes, [to] pass along the freedom to further copy and

require consumers who use OS software to license any improvements or extensions on similarly open terms.

This Part presents a framework for deciding when viral license terms do and do not violate the antitrust laws. We begin by reviewing judges' authority to strike down unnecessarily broad licenses under the Sherman Act. We then explore how viral licenses restrain (and in some cases also facilitate) competition. Next, we discuss various popular licenses that OS collaborations have used in the past. In the process, we find evidence that GPL and other so-called strong viral licenses are unnecessarily broad and that even some weaker licenses (for example, Mozilla) should often be viewed with suspicion. We close by identifying circumstances in which companies are most likely to adopt viral licenses as a deliberate cartelization strategy.

A. Antitrust Law and Restrictive Licenses

Many and perhaps most OS collaborations can be implemented in ways that satisfy the rule of reason.[148] However, this does not immunize every "naked restraint" (more precisely: "unnecessary output limiting appendage") that OS members decide to include in their licenses.[149] Indeed, such terms are per se illegal.[150] Do viral terms fit this definition? There is not much doubt that their announced purpose—encouraging companies that would normally create CS products to join OS instead—reinforces the cartel effect and therefore is "output limiting." The harder question is whether such restraints are "necessary." We will argue below that weak viral terms may sometimes be needed to stabilize OS collaborations in the first place. This is probably enough to escape per se illegality under current law.[151] On the other hand, judges must still conduct a rule of reason

change it." *What is Copyleft?*, GNU OPERATING SYSTEM, http://www.gnu.org/copyleft/ (last visited Mar. 11, 2012). Although originally pejorative, the "viral" label has largely lost this connotation through indiscriminate usage. This Article uses the term purely for its descriptive value. Readers interested in the ongoing semantic debate should consult WIKIPEDIA. *See Talk: Viral License*, WIKIPEDIA, http://www.en.wikipedia.org/wiki/Talk:Viral_license (last modified Dec. 7, 2011).

[148] *See supra* Part IV.

[149] 11 HERBERT HOVENKAMP, ANTITRUST LAW: AN ANALYSIS OF ANTITRUST PRINCIPLES AND THEIR APPLICATION ¶ 1906, at 267–70 (3d ed. 2011).

[150] *Id.*; Mark A. Lemley & Christopher R. Leslie, *Categorical Analysis in Antitrust Jurisprudence*, 93 IOWA L. REV. 1207, 1221 (2008).

[151] Doctrinally, the existence of a potential justification transforms the viral terms from a "naked restraint" to an "ancillary agreement." The threshold for this transformation is still unclear. As Professor Piraino has emphasized, courts are divided over whether restraints must be "plausibly related to the venture's procompetitive effects, reasonably related to those effects, or essential to achieving the effects." Thomas A. Piraino, Jr., *A Proposed Antitrust Approach to Collaborations Among Competitors*, 86 IOWA L. REV. 1137, 1189 (2001). Piraino argues that defendants should not be asked to prove "that a restraint was so essential that, but for the restraint, the venture could not operate at all" but only that "in the opinion of the joint venture partners, [the restraint was] reasonably

inquiry to see whether the benefits of OS sharing could be achieved by adopting some less restrictive alternative.[152] I argue below that such alternatives often exist, particularly for strong, GPL-type licenses.

Finally, suppose that a court finds that a particular viral license is overbroad. What relief can it grant? Unlike most Sherman Act cases, simple injunctive relief ordering parties "to cancel, shorten, or modify outstanding agreements" among two or more defendants[153] may be impractical. This is because the number of OS users will often be so large so that courts cannot identify, much less assert jurisdiction over every licensee. This is unlikely to be problematic for the vast majority of OS licenses that give named "stewards" the power to issue amended terms.[154] Here, it should normally be sufficient for the court to assert jurisdiction over the steward and order him or her to issue a less restrictive alternative license. For licenses without stewards, courts may have to content themselves with a declaration that the existing viral license violates the antitrust laws. This should allow users to assert a copyright misuse defense against anyone who tries to enforce the original license.[155]

B. Viral Economics

The Free Software Foundation ("FSF") invented GPL to encourage (or less politely, to compel) programmers to switch from CS to OS licenses.[156] In its

---

necessary for their success." *Id.* at 1190. This lower, subjective standard would make it much harder to strike down relatively weak viral licenses like Mozilla.

[152] HOVENKAMP, *supra* note 149, ¶ 1913, at 373–75 (citing Sullivan v. Nat'l Football League, 34 F.3d 1091, 1112 (1st Cir. 1994); Wilk v. Am. Med. Ass'n, 895 F.2d 352, 378 (7th Cir. 1990)).

[153] 2A PHILLIP E. AREEDA, ROGER D. BLAIR & HERBERT HOVENKAMP, ANTITRUST LAW: AN ANALYSIS OF ANTITRUST PRINCIPLES AND THEIR APPLICATION ¶ 325, at 5, 7 (3d ed. 2000).

[154] Most moderate and strong viral clauses include steward clauses. Examples include the Free Software Foundation's GPLv2, *see infra* note 177, ¶ 9; GPLv3, *see infra* note 185, ¶ 14; and LGPL, *see infra* note 165, ¶ 6; Netscape's Mozilla License, *see infra* note 198, ¶ 6; Apple Computer's Public Source License, *see infra* note 204, ¶ 7; Sun's Sun Industry Standards License, *see infra* note 215, ¶ 6; the IBM Public License, *see infra* note 207, ¶ 7; the IBM Common Public License, *see infra* note 208, ¶ 7; and the Eclipse Public License, *see infra* note 209, ¶ 7. Among the licenses reviewed for this article, only the very minimal Apache License, *see infra* note 221, and licenses modeled on the Berkeley Software Distribution license, *see infra* note 224, fail to name a steward.

[155] *See* Ilan Charnelle, *The Justification and Scope of the Copyright Misuse Doctrine and Its Independence of the Antitrust Laws*, 9 UCLA ENT. L. REV. 167, 167–68 (2002); Brett Frischmann & Dan Moyland, *The Evolving Common Law Doctrine of the Copyright Misuse: A Unified Theory and Its Application to Software*, 15 BERKELEY TECH. L.J. 865, 867–70 (2000); Dennis S. Karjala, *Copyright Protection of Operating Software, Copyright Misuse, and Antitrust*, 9 CORNELL J.L. & PUB. POL'Y 161, 165–69 (1999).

[156] *See* K.S.SAMPATHKUMAR, UNDERSTANDING FOSS VERSION 4.0N: FREE OPEN SOURCE SOFTWARE 2–4 (2010).

original form, FSF's argument was based on the assumption that GPL would be used to license "unique" OS modules that no CS program could match.[157] Programmers who wanted to use the modules would therefore have to adopt GPL themselves. This central assumption that a module could be "unique" was probably reasonable in a world dominated by small individual programmers who lacked the resources to duplicate large code bases. However, it was always more doubtful for commercial developers[158] who, as FSF never tired of pointing out, had "the advantage of money."[159] For companies, the question was less whether a supposedly "unique" program could be duplicated—clearly it could[160]—but whether the profits from doing so were greater than those that could be earned by adopting GPL.

The rise of commercial OS, then, has rendered FSF's original "uniqueness" argument obsolete. Even so, the basic instinct is sound. To see why, consider how a company would go about making OS/CS decisions for a group of proposed software modules. Following Part III, a CEO would start by estimating the relative profitability of developing each module using OS and CS methods. This would allow her to sort the modules into four categories:

(I) OS is much more profitable than CS;
(II) OS is slightly more profitable than CS;
(III) OS is slightly less profitable than CS; and
(IV) OS is much less profitable than CS.

At this point our CEO's investment decision would depend on how the OS license was structured. A traditional, nonviral license would let the company make separate module-by-module choices. In this case, we would expect our CEO to develop the Category I and II modules as OS and the Category III and IV modules as CS. Viral licenses change this result by requiring the company to make take-it-

[157] FSF based its argument on experience with a program called "GNU Readline" which reportedly had "a significant unique capability." FSF believed that "Releasing [Readline] under the GPL and limiting its use to free programs [gave] our community a real boost. At least one application program is free software today specifically because it was necessary for using Readline." Richard Stallman, *Why You Shouldn't Use the Library GPL for Your Next Library*, GNU OPERATING SYSTEM (FEB. 1999), http://www.gnu.org/software/chinese/www/why-not-lgpl.2002-09-26.html.

[158] FSF seems to have assumed that commercial companies would continue to practice CS long after most individual programmers adopted OS. This explains its curiously muted boast that "Nowadays, as companies *begin to consider* making software free, *even some* commercial projects can be influenced in this way." *Id.* (emphasis added).

[159] *Id.*

[160] Because duplication takes time, it remains possible that some OS modules could be temporarily unique. However, this probably does not happen very often. The reason is that temporary uniqueness implies a head start, that is, that CS companies only learn about the OS module after it has been written. Today's companies monitor OS developments so closely that such surprises seem unlikely.

or-leave-it decisions for bundles containing multiple modules.[161] Here, shrewd license design can increase the number of modules that the company develops in OS mode by, for example, bundling some Category III and IV modules with a much larger number of Category I and II modules.[162] The trick, of course, is to make sure that the former outnumber the latter. Viral licenses that include too many Category III and IV modules could easily backfire by persuading our CEO that CS development was more lucrative after all.

In the real world, designing licenses with just the right infectiveness is a rough-and-ready business.[163] GPL solves this problem by using objective proxies based on the extent to which new modules copy or dynamically link to preexisting OS code.[164] Specifying multiple or unusual links (a) limits the number of Category III and IV modules that companies are asked to embrace, and (b) provides at least some evidence that the module's function is similar to the underlying OS software and therefore more likely to fall within Category III than Category IV. On the other hand, FSF acknowledges that these proxies can sometimes be wrong. It therefore encourages OS programmers to use the so-called Lesser General Public License ("LGPL")[165] in cases where the normal GPL license would be ineffective or counterproductive. Unlike the GPL, the LGPL permits dynamic linking which makes it markedly less infective.[166]

Suppose, then, that viral terms succeed in persuading companies to adopt OS more often. What are the downsides? First and most obviously, viral licenses increase the number of companies that choose OS development for any given module.[167] This aggravates the already large imbalance between OS and CS companies and reinforces the cartel effect. Second, software development is not an

[161] In principle, OS architectures can also be manipulated to change the boundaries between modules. This is because programmers have considerable technical freedom to place software in one "container" instead of another. Greg R. Vetter, *"Infectious" Open Source Software: Spreading Incentives or Promoting Resistance?*, 36 RUTGERS L.J. 53, 101–13 (2004). This means that it should be possible to gerrymander software projects so that the functions that companies prefer to develop using CS methods are concentrated in as few modules as possible. Such a strategy would pose significant antitrust problems given courts' admitted reluctance to second-guess software designers' architecture choices. *See, e.g.*, United States v. Microsoft Corp., 253 F.3d 34, 84 (D.C. Cir. 2001) (expressing reluctance to second-guess engineering decision to combine previously separate software programs).

[162] Category III modules may, however, be developed less robustly than they would be under a CS business plan.

[163] *See* Lerner & Tirole, *supra* note 5, at 202–04.

[164] Applications programs are said to use "static links" when code from preexisting software is copied and permanently incorporated during installation. By contrast, "dynamic links" install code temporarily while the application is running and later erase it again. Vetter, *supra* note 161, at 82–85, 104–06.

[165] *See GNU Lesser General Public License Version 3.0*, GNU OPERATING SYSTEM (June 29, 2007), http://www.gnu.org/copyleft/lesser.html.

[166] *See* Vetter, *supra* note 161, at 104–05, 199.

[167] *See supra* Part V.A.

all-or-nothing process. Companies that would have preferred to use CS methods will normally spend less if viral terms force them to use OS methods instead. This suggests a Faustian bargain: broad viral terms produce more OS software, but this gain is more than offset by lost CS production.[168]

C. *Justifications*

If viral terms are the problem, why not get rid of them? Unfortunately, things aren't so simple. The reason is that some viral terms may still be needed to keep collaborations from unraveling. To see why, consider a company that would like to improve or extend an existing OS module. Plainly, its most profitable business strategy is to copy the underlying OS module at zero cost and then sell CS improvements for whatever the market will bear. This, of course, is a formula for disaster. If every company does this, OS will disappear entirely.[169]

Viral provisions block this outcome by forcing companies that use OS code to donate their improvements back to the community.[170] The question is, how narrow can this restriction be and still ensure stability? [171] Logically, viral terms should be at least as broad as the underlying OS module. On the other hand, this might not be enough. The problem, as Professor Vetter has noted, is that programmers have considerable technical freedom to move software from one module to another.[172] This means that viral clauses may have to be broader than the original OS module to enforce stability. Probably the simplest solution is to extend the viral term to any module that includes code copied from an underlying OS program. This approach is found in the widely used Mozilla license and its imitators.[173] This may not be sufficient, however, in cases where companies can write a second module that links to the first module and causes it to behave as if it had been rewritten. In such cases, the viral provision must cover *both* the first module *and* at least some of the modules that link to it.[174]

[168] *See supra* Part V.A; *see also* Vetter, *supra* note 161, at 152–53.

[169] Suzanne Scotchmer, *Openness, Open Source, and the Veil of Ignorance*, 100 AM. ECON. REV., May 2010 at 169–70; von Engelhardt & Maurer, *supra* note 59, at 10.

[170] Vetter, *supra* note 161, at 155–56 n.272.

[171] Professor Katz has noted a similar problem in the context of joint R&D ventures. He warns that "[a]ntitrust authorities should be wary of agreements that attempt to limit the R&D that a member firm may conduct without sharing. Firms should have to prove that the restraint is essential to the proper functioning of the cooperative agreement." Katz, *supra* note 76, at 542.

[172] Vetter, *supra* note 161, at 101–13.

[173] *See Mozilla Public License Version 2*, MOZILLA.ORG, http://www.mozilla.org/MPL/2.0/ (last visited Jan. 16, 2012).

[174] The license does not have to proscribe all or even most links. It only has to narrow the programmer's toolbox of possible links to the point where evading OS status is no longer worth the trouble. This is similar to the usual argument that the market power conferred by patents is measured by the "inventing around" costs. Scotchmer, *supra* note 68, at 105, 107–12.

That said, the argument for viral provisions contains an important loophole. We have assumed that companies can sell their CS improvements on the open market.[175] This may not be true for a variety of reasons. First, the modifications may be so specialized or have such limited value that finding potential buyers is not worth the transaction cost. Second, consumers may find it prohibitively expensive to judge software quality for themselves. In this case, they may prefer OS software, not because it is better, but because the collaboration has endorsed it. Finally, consumers may worry that CS software could stop working if and when the "official" software changes. In any of these cases, viral terms could be completely unnecessary.

This is as far as theory can take us. At this point, proving the existence of less restrictive alternatives becomes an empirical question. As Professor Hovenkamp has remarked:

> [P]laintiffs cannot be permitted to offer possible less restrictive alternatives whose efficacy is mainly a matter of speculation. . . . Proffered less restrictive alternatives should either be based on actual experience in analogous situations elsewhere or else be fairly obvious. Tending to defeat such an offering would be the defendant's evidence that the proffered alternative has been tried but failed, that it is equally or more restrictive, or otherwise unlawful.[176]

Fortunately, commercial OS collaborations have used many different viral licenses over the past decade. We now turn to this evidence.

1. *Case 1: Strong Viral Terms*

The broadest and most stringent viral terms are invariably found in FSF's GPL licenses and a handful of imitators. These include:

> *GPLv2.* This is the best-known and strongest viral license. Formally, it extends to any software "based on" protected code.[177] While the scope of this phrase has been much debated,[178] it seems to include (a) files that include any verbatim portion of the preexisting code,[179] (b) entirely new modifications to that code,[180] (c) any interface files, compiler and

[175] *See supra* text accompanying note 169.

[176] HOVENKAMP, *supra* note 149, at 375–76.

[177] *GNU General Public License Version 2*, GNU OPERATING SYSTEM, para. 0 (June 1991), http://www.gnu.org/licenses/old-licenses/gpl-2.0.html (last visited Jan. 15, 2012) [hereinafter GPLv2].

[178] *See, e.g.*, Jason B. Wacha, *Taking the Case: Is the GPL Enforceable?*, 21 SANTA CLARA COMPUTER & HIGH TECH. L.J. 451, 487 (2005) (GPL's "based on" language presents "a thorny question of interpretation").

[179] GPLv2, *supra* note 177, at para. 0.

[180] *Id.*

installation scripts needed to run that code,[181] and (d) all "derivative work,"[182] a term that arguably incorporates the full reach of "derivative works" under US copyright law.[183] The concept of verbatim copies is also said to include both static links that copy software prior to use and dynamic links which make transient copies while the program is running. FSF argues that the license's scope should normally be evaluated based on the number and type of interactions between the underlying OS software and any new software.[184] To the extent that viral terms apply, programmers must adopt GPLv2 for their own work and cannot charge royalties.[185]

GPLv3. This license was designed to simplify GPLv2's sometimes obscure "based on" language. This is done by extending the viral term to any software that "cop[ies] from" or "adapt[s] all or any part" of the underlying OS software.[186] Programmers who write such software must license it at zero royalties under GPLv3[187] and also supply enough "corresponding source" software for users to run the code.[188]

Oracle's Berkeley Database License.[189] The scope of this license includes both preexisting OS code and "modifications."[190] Programmers whose work falls within this definition must make their source code

[181] *Id.* at para. 3. The license contains a "special exception" for "anything that is normally distributed . . . with the major components . . . of the operating system on which the executable runs . . . ." *Id.*

[182] *Id.* at para. 0.

[183] GPLv2 does create an exception for code "written entirely by you." *Id.* at para. 2. However, this is implemented in a fairly minimal way by excluding software only when it is distributed as a "separate work." *Id.* GPLv2 also creates exceptions for software that is part of a "mere aggregation" on a storage medium. *Id.*

[184] FSF has suggested that courts should decide which links qualify based on both "the mechanism of communication (exec, pipes, rpc [*sic*], function calls within a shared address space, etc.) and the semantics of the communication (what kinds of information are interchanged)." Ron Phillips, *Deadly Combinations: A Framework for Analyzing the GPL's Viral Effect*, 25 J. MARSHALL J. COMPUTER & INFO. L. 487, 493–94 (2008).

[185] GPLv2, *supra* note 177, at paras. 10, 14.

[186] *GNU General Public License Version 3*, GNU OPERATING SYSTEM, para. 0, http://www.gnu.org/licenses/gpl-3.0.html (last visited Jan. 15, 2012).

[187] *Id.* at para. 10.

[188] *Id.* at para. 6. Significantly, the term excludes "system libraries" and "general purpose" tools. *Id.* at para. 1. This presumably reflects a realization that companies faced with such demands would reject GPL entirely.

[189] *Open Source License for Oracle Berkeley DB*, ORACLE.COM, http:// www.oracle.com/technetwork/database/berkeleydb/downloads/oslicense-093458.html (last visited Jan. 18, 2012).

[190] *Id.*

available to users together with "any accompanying software that uses" it.[191]

There are several reasons to think that GPL and its progeny are much broader than stability requires. First, FSF designed GPL to have the maximum possible impact. Nothing in the written record suggests that FSF thought that viral clauses could be too strong, let alone that the ratio of OS to CS companies could ever be too high.[192] Second, FSF knew that narrower licenses were possible and stable. The existence of an LGPL that lets CS programs dynamically link to OS libraries proves this.[193] Finally, Linus Torvalds has repeatedly relaxed GPL so that programmers can write CS programs that make "normal system calls" on LINUX[194] and even insert CS "loadable kernels" into LINUX itself.[195] Despite this, the LINUX collaboration shows no signs of instability.

2. Case 2: Weak Viral Terms

LINUX apart, "infectious terms have been GPL's least imitated feature."[196] Probably the biggest reason is that broad viral terms make it harder for software companies to earn revenue from complementary CS programs. Relaxing this restriction strengthens OS collaborations by allowing more companies to participate.[197] This may explain why commercial OS collaborations overwhelmingly choose so-called "weak copyleft licenses." Examples include:

---

[191] *Id.* at para. 3. The license is tempered by Oracle's statement that it is prepared to negotiate royalty-bearing licenses that waive these obligations. *See id.* This suggests that—unlike FSF's licenses—the Oracle license is mainly a negotiating ploy.

[192] *See, e.g.*, Stallman, *supra* note 157.

[193] Some commentators have also argued, contrary to FSF, that GPLv.2 also permits dynamic links. *See, e.g.*, Lothar Determann, *Dangerous Liaisons— Software Combinations as Derivative Works? Distribution, Installation, and Execution of Linked Programs Under Copyright Law, Commercial Licenses, and the GPL*, 21 BERKELEY TECH. L.J. 1421, at 1491, 1496 (2006).

[194] Greg R. Vetter, *Claiming Copyleft in Open Source Software: What if the Free Software Foundation's General Public License (GPL) Had Been Patented?*, 2008 MICH. ST. L. REV. 279, 311–12. The situation has been muddied by Torvalds' suggestion that Linux *does* extend to software that is "so obviously Linux-specific that it simply wouldn't make sense without the Linux kernel." Mitchell L. Stoltz, *The Penguin Paradox: How the Scope of Derivative Works in Copyright Affects the Effectiveness of the GNU GPL*, 85 B.U. L. REV. 1439, 1452–53 (2005). Conversely, Torvalds has argued that GPL does not reach preexisting software that has been modified to place calls on the Linux kernel. Vetter, *supra* note 161, at 154–55.

[195] John Tsai, Note, *For Better or Worse: Introducing the GNU General Public License Version 3*, 23 BERKELEY TECH. L.J. 547, 559–60 (2008).

[196] Vetter, *supra* note 161, at 151.

[197] For example, Sun decided not to use GPL because it wanted collaboration members to be able to create "larger works for commercial purposes." *FAQ: Common Development and Distribution License (CDDL)*, OPEN SOLARIS, (Nov. 10, 2011, 3:38 PM),

Mozilla Licenses. The "Mozilla Public License"[198] covers all underlying OS code and "modifications." The latter are defined to include (a) additions or deletions from existing files, and (b) any new module that includes copies of licensed code.[199] In practice, it is usually interpreted in ways that permit CS programs to include unlimited dynamic and static links to the underlying software and also construct "larger" CS works that bundle OS and CS software together.[200] Covered software must, however, be made available to users as source code on a royalty-free basis.[201] Sun's "Common Development and Distribution License"[202] and "Sun Public License"[203] contain similar provisions.

Apple Public Source License.[204] This license asserts Mozilla-type breadth but permits companies to sell CS modifications provided that the source code is supplied to users.[205] Here, the formal ability to license extensions is undercut by the requirement to distribute source code. This makes it easier for consumers to evade licensing restrictions on, for example, transferring the software to unauthorized users. Similar requirements can also be found in Sybase's "Sybase Open Source License."[206]

IBM/Eclipse Licenses. The "IBM Public License,"[207] "IBM Common Public License,[208] and Eclipse Foundation "Eclipse Public License"[209] all

http://hub.opensolaris.org/bin/view/Main/licensing_faq#HCanIredistributeorselltheOpenSolarissourcec.

[198] *Mozilla Public License Version 2.0*, MOZILLA.ORG, para. 1.10, http://www.mozilla.org/MPL/2.0/ (last visited Feb. 6, 2012).

[199] *Id.* at para.1.9.

[200] *See, e.g.*, *GNU General Public License*, WIKIPEDIA, http://en.wikipedia.org/wiki/GNU_General_Public_License (last visited Jan. 16, 2012).

[201] *Mozilla Public License Version 2.0*, *supra* note 198 at para. 2.1.

[202] *Common Development And Distribution License Version 1.0*, OPEN SOURCE INITIATIVE, http://www.opensource.org/licenses/cddl1.php (last visited Jan. 18, 2012).

[203] *Sun Public License Version 1.0*, ORACLE, http://java.sun.com/spl.html (last visited Jan. 18, 2012).

[204] *Apple Public Source License Version 2.0*, APPLE.COM (Aug. 6, 2003), http://www.opensource.apple.com/license/apsl/.

[205] *Id.* at para. 2.2(c).

[206] *The Sybase Open Watcom Public License 1.0*, OPEN SOURCE INITIATIVE, para. 2.2(c), http://www.opensource.org./licenses/sybase.php (last visited Jan. 18, 2012).

[207] *IBM Public License Version 1.0*, OPEN SOURCE INITIATIVE, http://www.opensource.org/licenses/ibmpl.php (last visited Jan. 18, 2012).

[208] *Common Public License (CPL) -- V1.0*, IBM, http://www.ibm.com/developerworks/library/os-cpl.html (last visited Jan. 18, 2012). The license's viral terms are limited to "Contributions" to the program. *Id.* at para. 1. They specifically exclude

assert Mozilla-type breadth.[210] However, they are much less stringent since they let users sell the underlying OS code (with or without modifications) under CS licenses provided that they make their source code available.[211] Furthermore, the Eclipse license does not reach "separate modules" unless they qualify as "derivative works" under US copyright law.[212] While Eclipse concedes that the precise boundaries of this exception are unclear,[213] companies are clearly permitted to write CS licenses for Eclipse plug-ins that contain 100 percent new code and "implement functionality not currently in Eclipse."[214]

Sun Industry Standards Source License (SISSL).[215] Sun provided its Open Office software under this license until 2006.[216] The unusual document asserts Mozilla-type breadth, but permits programmers considerable freedom to write CS software. Thus, they can sell modifications under CS licenses provided that (a) they continue to make the original source available under the SISSL license, and (b) their modifications meet Open Office standards. Conversely, companies whose products fail to satisfy Open Office standards must publish a list of all deviations and/or the modification's source code.[217] In either case, the obligation to disclose source code is much lower than for other Mozilla-type licenses.

---

software that (i) is found in separate modules distributed in conjunction with the Program under a separate license agreement, and (ii) is not a derivative work of the Program. *Id.*

[209] *Eclipse Public License - v 1.0*, ECLIPSE FOUND., http://www.eclipse.org/legal/epl-v10.html (last visited Jan. 18, 2012).

[210] *See IBM Public License Versiòn 1.0*, *supra* note 207, at para. 1; *Common Public License (CPL) -- V1.0*, *supra* note 208, at para. 1; *Eclipse Public License - v 1.0*, *supra* note 209, at para. 1.

[211] *IBM Public License Version 1.0*, *supra* note 207, at para. 3(b)(iv); *Common Public License (CPL) -- V1.0*, *supra* note 208, at para. 3(b)(iv); *Eclipse Public License - v 1.0*, *supra* note 209, at para. 3(b)(iv).

[212] *IBM Public License Version 1.0*, *supra* note 207, at para. 1(b)(ii); *Common Public License (CPL) -- V1.0*, *supra* note 208, at para. 1(b)(ii); *Eclipse Public License - v 1.0*, *supra* note 209, at para. 1(b)(ii).

[213] For example, the Foundation does not say how many links to, or copying of the underlying OS software triggers "derivative work" status. *Eclipse Public License (EPL) Frequently Asked Questions*, ECLIPSE FOUND., para. 26, http://www.eclipse.org/legal/eplfaq.php (last visited Jan. 18, 2012).

[214] *Id.* at para. 27.

[215] *Sun Industry Standards Source License - Version 1.1*, OPENOFFICE.ORG, http://www.openoffice.org/licenses/sissl_license.html (last visited Jan. 18, 2012).

[216] Open Office moved to LGPL in 2005. Steven Vaughan-Nichols, *Sun Changes OpenOffice.org Licensing*, EWEEK.COM (Sept. 6, 2009), http://www.eweek.com/c/a/Linux-and-Open-Source/Sun-Changes-OpenOfficeorg-Licensing/ (last visited Jan. 18, 2012).

[217] *Sun Industry Standards Source License - Version 1.1*, *supra* note 215, at para. 3.1.

The popularity of the foregoing licenses provides powerful evidence that even relatively narrow viral terms are enough to stabilize OS collaborations. First, none of the licenses ban CS licenses that link to preexisting code. Indeed, the IBM and Eclipse licenses grant companies an unrestricted right to sell separate CS "plug-in" modules provided that they contain entirely new software.[218] Second, only the Mozilla license completely bars users from selling modified code under CS licenses.[219] That said, all of the above-listed non-Mozilla licenses also restrict this right, usually by requiring users to make the source code for their modifications available to users.[220] Because these terms make CS licensing more difficult, they are at least arguably related to stability.

3. *Case 3: Minimally Viral Terms*

Finally, many commercial projects contain only minimal viral terms. Examples include:

> *Apache License.*[221] This license formally reaches all software that is based on or derived from the underlying OS code.[222] However, it only requires licensees to reproduce the preexisting software's copyright, trademark, attribution, and legal notices. Users are otherwise free to sell modified (and even original) versions of the underlying software under CS licenses.[223]
>
> *Berkeley Software Distribution (BSD).* BSD formally covers all "redistribution with or without modification."[224] However, licensees' obligations are limited to retaining certain disclaimers and notices.[225] This gives users an unrestricted right to license both original and

[218] IBM Public License Version 1.0, *supra* note 207, at para. 2; Common Public License (CPL) -- V1.0, *supra* note 208, at para. 2; Eclipse Public License - v 1.0, *supra* note 209, at para. 2.

[219] *Mozilla Public License Version 1.1*, *supra* note 198, at para. 2.1.

[220] Requiring users to disclose source code for their modifications almost certainly makes CS less attractive. Given a choice, practically all commercial CS companies withhold source code as a way of discouraging unauthorized use.

[221] *Apache License, Version 2.0*, APACHE SOFTWARE FOUNDATION (Jan. 2004), http://www.apache.org/licenses/LICENSE-2.0 (last visited Jan. 18, 2012); *see also Apache License and Distribution FAQ*, APACHE SOFTWARE FOUNDATION, http://www.apache.org/foundation/licence-FAQ.html#My-License (last visited Jan. 18, 2012).

[222] *Apache License, Version 2.0*, *supra* note 221, at para. 1. The license gives no indication that it seeks to incorporate the meaning ascribed to "derivative works" under the US Copyright Act.

[223] *Id.* at para. 4. Programmers must, however, offer royalty-free licenses on any software which they donate to the Apache Foundation itself. *See id.* at 5.

[224] *The BSD License*, OPEN SOURCE INITIATIVE, http://www.opensource.org/licenses/bsd-license.php (last visited Jan. 18, 2012).

[225] *See id.*

modified versions of the underlying OS software commercially. Commercial versions of BSD include the "Cryptix General License"[226] and the "Intel Open Source License."[227]

The popularity of Apache and BSD licenses shows that many OS collaborations can be stabilized with minimal viral restrictions. That said, there may be cases where weakly viral, Mozilla-type licenses are necessary. The reason is that the Apache and BSD licenses usually appear in markets where one-man businesses deliver custom software to individual clients. Here, customers find it impractical to reuse or share software no matter what their license says. It would be more informative to find Apache or BSD-style licenses in mass markets. Apple's decision to make many of its software products available under Apache instead of the weakly viral Apple Public Source license[228] suggests that such markets do, in fact, exist.

D. *Viral Licenses in a Commercial Age*

We have argued that viral terms—and especially strong, GPL-type licenses—needlessly extend cartel effects and limit the amount that companies invest in OS. Historically, these terms have usually been motivated by ideology.[229] However, we have seen that companies can also use such licenses to reinforce the cartel effect. The appeal of this strategy will normally depend on the type of company involved:

Software Companies. We have already emphasized that OS modules are most profitable when CS competition is weak. GPL terms provide a natural way to suppress CS and make these modules even more profitable. The main constraint, as we have seen, is that most software companies would prefer not to cartelize the Killer App market. GPL licenses that do this are unlikely to be adopted.

Hardware and Service Providers. Hardware and service providers face a more ambiguous choice. On the one hand, strong viral terms increase the supply of free software. This lets consumers spend more of their IT budgets on hardware and services. On the other hand, we have seen that strong viral terms suppress CS competition. This will normally reduce

[226] *Cryptix General License*, CRYPTIX.ORG, http://www.cryptix.org/LICENSE.TXT (last visited Jan. 18, 2012).

[227] *The Intel Open Source License*, OPEN SOURCE INITIATIVE, http://www.opensource.org/licenses/intel-open-source-license.php (last visited Jan. 18, 2012).

[228] *Apple Public Source License*, WIKIPEDIA, http://en.wikipedia.org/wiki/Apple_Public_Source_License (last visited Jan. 17, 2012).

[229] *But see* Josh Lerner & Jean Tirole, *The Scope of Open Source Licensing*, 21 J.L. ECON. & ORG. 20, 53–55 (2005) (arguing that commercial firms may sometimes adopt more restrictive OS licenses to attract volunteer programmers who would not otherwise participate).

> the total supply of software so that consumers receive less value from whatever hardware and services they do buy. For this reason, the decision to adopt strong, GPL-style licenses will usually be highly fact dependent.

For most businesses, then, viral terms offer both benefits and costs. This may explain why most commercial OS collaborations use weaker licenses.[230] At the same time, deliberate cartelization remains a plausible threat. This provides yet another reason for courts and policymakers to scrutinize GPL-style licenses.

VI. GOVERNMENT INNOVATION POLICY: TAXATION, SUBSIDIES, GRANTS, AND PROCUREMENT POLICY

Part III has argued that the OS cartel effect suppresses Year One competition and leads to a systematic imbalance between OS and CS companies. On the other hand, Part IV suggests that these costs will often be acceptable under the rule of reason. At this point, nothing the Sherman Act can do to protect Year Two competition is likely to fix the imbalance. Relief, if it comes at all, will require government intervention.

The idea of government intervention has been widely debated for most of the past decade.[231] At first, pro-OS scholars assumed a simple objective: if free software was good, then more software would be better.[232] This led to many different proposals for promoting OS through procurement preferences, taxation, subsidies, and other interventions.[233] Significantly, scholars who opposed these suggestions seldom argued with the assumption that more OS was always desirable. Instead, they raised a practical objection. Given that OS volunteers were

---

[230] *See supra* notes 196–220 and accompanying text.

[231] *Compare* LAWRENCE LESSIG, THE FUTURE OF IDEAS 247–49 (2001) (arguing for more government encouragement of OS solutions), *with* David S. Evans & Bernard J. Reddy, *Government Preferences for Promoting Open-Source Software: A Solution in Search of a Problem*, 9 MICH. TELECOMM. & TECH. L. REV. 313, 393–94 (2003) (disagreeing, and arguing that governments should not intervene in favor of OS solutions where there is "no general market failure . . . in the provision of commercial software").

[232] *See* LESSIG, *supra* note 231, at 174–76 (arguing that the benefits of freer Internet infrastructure and code substantially outweigh the benefit of corporate controlled infrastructures).

[233] *See, e.g.*, *id.* at 247; LERNER & SCHANKERMAN, *supra* note 16, at 199 ("Incentives for software development or adoption can take a variety of forms, including monetary incentives such as direct subsidies and tax credits. But they can also be more indirect, such as when governments attach specific software conditions to procurement decisions, regulatory policy and even informal pressure and 'moral suasion.'"). Other scholars have called for "introducing a National Software Foundation . . . that will fund software development projects on condition that the fruits be licensed as free software, and the adoption of a . . . policy that would require that software written under government contract be released as free software." Yochai Benkler, *Freedom in the Commons: Towards a Political Economy of Information*, 52 DUKE L.J. 1245, 1275 (2003).

hardly ever motivated by money,[234] traditional policy levers based on taxation or spending were unlikely to change behavior.[235] This academic discussion was followed by a similarly inconclusive debate in governments around the world.[236] While some promotion schemes were enacted, most were abandoned and new proposals dropped off rapidly after 2005.[237] This does not, however, mean that governments lost interest. Instead, the original idea that there could never be enough OS became more nuanced. In the words of one leading observer, schemes to promote OS have become "subsumed" in a broader "search for business models that can profitably blend open and proprietary processes and products."[238] But what these OS/CS blends might like look like was seldom, if ever, specified.

The rise of commercial OS changes—and significantly clarifies—this debate. On the one hand, the old objection that government cannot possibly influence an activity motivated by "fun" no longer holds. Modern commercial OS clearly responds to financial incentives.[239] On the other hand, the cartel effect provides a much clearer analytical framework for deciding when the "right" OS/CS blend has been achieved. This section asks how government can use the various options at its disposal to suppress the cartel effect and help OS sharing reach its full potential.

[234] Evans & Reddy, *supra* note 231, at 341, 344 (OS has "primarily been developed by individuals who donate their time to work on projects that interest them" and "[t]he circumstances under which a for-profit firm has incentives to invest in open source, particularly under the GPL, may be limited."); Jyh-An Lee, *New Perspectives on Public Goods Production: Policy Implications of Open Source Software*, 9 VAND. J. ENT. & TECH. L. 45, 47 (2006) (profit incentive "does not exist in the open source community"); Klaus M. Schmidt & Monika Schnitzer, *Public Subsidies for Open Source? Some Economic Policy Issues of the Software Market*, 16 HARV. J.L. & TECH. 473, 481–84 (2003) (noting that most OS is motivated by nonmonetary incentives and asserting that commercial OS investments "are likely to remain limited" because of free-riding).

[235] Evans & Reddy, *supra* note 231, at 389; Schmidt and Schnitzer, *supra* note 234, at 498–99.

[236] Politicians around the globe have given various reasons for promoting OS including favoring local industry (Argentina, Australia, Brazil, China, Costa Rica, Germany), putting pressure on CS companies to compete (Germany, Denmark, Taiwan, Thailand), becoming less dependent on American multinationals (Costa Rica, Japan, Russia, Thailand), improving computer security (Costa Rica, Argentina, Iran) and reducing copyright piracy (Iran, Thailand). *Government Open Source Policies*, CENTER FOR STRATEGIC & INT'L STUD. (July 2008), www.csis.org/files/media/csis/pubs/0807218_government_opensource_policies.pdf; *see also* Lee, *supra* note 234, at 101, 105.

[237] *E.g.*, *Government Open Source Policies*, *supra* note 236.

[238] *Id.* at 1.

[239] *See supra* notes 12–15 and accompanying text.

A. Purchasing Preferences

Governments already spend large amounts of money purchasing software.[240] For this reason, politicians often argue that redirecting these resources to promote OS is essentially costless.[241] Since the early 2000s, at least sixteen countries have considered legislation or regulation that would require government agencies (and in some cases state-owned companies) to adopt OS solutions when such products exist.[242] Furthermore, at least four countries have actually adopted such measures[243] and six more have adopted "preferences" for OS where its performance is comparable to CS alternatives.[244]

Do these policies make sense? It depends. We have seen that market imperfections can (a) prevent all-OS markets from evolving into mixed OS/CS states,[245] (b) prevent all-CS markets from evolving into mixed OS/CS states,[246] and (c) systematically over-supply OS companies where mixed OS/CS states already exist.[247] Using pro-OS purchasing preferences to encourage new OS entry would be irrelevant in case (a) but potentially helpful in case (b). On the other hand, we have seen that there are already too many OS companies in mixed markets.[248] This

[240] Professors Lerner and Schankerman's survey of 1,894 companies operating in fifteen countries found that government purchases accounted for 21 percent of total revenue. LERNER & SCHANKERMAN, *supra* note 16, at 177. This figure was "about equally divided" among federal, local, and municipal governments. *Id.*

[241] *See, e.g.*, *Government Open Source Policies*, *supra* note 236, at 19.

[242] Argentina, Australia, Belgium, Brazil, Bulgaria, Chile, Colombia, Finland, Italy, Malaysia, The Netherlands, Peru, Portugal, Spain, Ukraine, and Venezuela have all considered bills or regulations to implement mandatory open source procurement. *Id.* at 3–6, 8, 11–16, 19.

[243] The Netherlands, Peru, South Africa, and Venezuela have all adopted some version of mandatory preferences at the national level. Many more countries have adopted mandatory adoption requirements at the state and local level. For example, many Brazilian state and municipal governments have established requirements or preferences in favor of OS. *Id.* at 4, 13–14, 16, 19.

[244] Australia, Belgium, Brazil, China, Malaysia and Spain. *Id.* at 3–5, 12, 16. This probably understates the number of countries that have adopted preferences. For example, France claims that it has merely "encouraged" OS use in government. However, its 96 percent adoption rate suggests that something more systematic is at work. Gijs Hillenius, *FR: Almost the Entire Public Sector is Using Open Source*, EUROPEAN COMMISSION JOINUP (Sept. 30, 2009), https://joinup.ec.europa.eu/news/fr-almost-entire-public-sector-using-open-source.

[245] *See supra* text accompanying notes 77–88.

[246] *See supra* text accompanying notes 86–91.

[247] *See supra* text accompanying notes 93–98.

[248] *See supra* text accompanying notes 93–98.

suggests that government procurement preferences for OS software would make problem (c) even worse than it is today.[249]

But if pro-OS preferences are a bad idea, why not adopt pro-CS preferences instead? Currently, the idea is politically unlikely. But this could change over time as the image of OS becomes more corporate. In the meantime, it is important to know whether a pro-CS procurement policy makes sense. Here the good news is that preferences would help CS companies win more business and become more profitable. This would eventually lead to more CS entrants and dilute the cartel effect. However these gains would come at a price:

1. Fine-Tuning

We have argued that government should try to achieve the right "blend" of OS and CS companies. But how will government know when it achieves this mix? Certainly, any idea of fine-tuning seems hopelessly optimistic. On the other hand, this may not matter if—as Figure 4 suggests—the current blend is far from ideal. Several scholars have pointed out that government purchases are too small to have more than a minor impact on software markets.[250] This could be a good thing if it means that there is no chance of overshooting the desired mix.

2. Crowding Out

By definition, procurement preferences let CS companies sell more software with identical effort or sell the same amount of software with less effort. In practice, we expect a mix of both. This suggests that government preferences will cause CS companies to increase their investments, but not as fast as the value of the preference.

3. Costs to Government

Government procurement preferences only matter to the extent that they force government to choose software that it would not otherwise use. We should therefore assume that government will receive less value (and fewer capabilities) for every dollar it spends on software.

[249] For a different argument that pro-OS subsidies are counterproductive, see LERNER & SCHANKERMAN, *supra* note 16, at 170 (arguing that OS code is not priced at its true cost and this artificially suppresses CS markets).

[250] Lee, *supra* note 234, at 60 n.86 (estimating $17 billion government procurement market worldwide as of 2002). *But see* LERNER & SCHANKERMAN, *supra* note 16, at 177 (suggesting that government purchases account for one-fifth of all software revenue).

B. Subsidies

Governments have launched a variety of initiatives to subsidize OS methods including grants and cash incentives to private sector OS adopters,[251] government-funded projects and alliances to promote OS methods,[252] government-funded training, support, and demonstration centers,[253] and government-funded OS workshops.[254] Since government sometimes promotes CS models as well—particularly in the case of start-ups—it is hard to know how much these policies favor OS on net.[255] Nevertheless, it is at least reasonable to think that OS firms receive substantially more support than their CS competitors.

One nice feature of these policies is that they are disproportionately aimed at helping OS companies penetrate new markets. This suggests that they may sometimes help all-CS markets avoid lock-in. The case is more difficult for subsidies in mixed markets where OS is already firmly established. Here, subsidies—like procurement policy—are likely to crowd out significant amounts of private investment.

Once again, the deepest problem relates to fine-turning—that is, knowing when to stop the subsidies once a mixed OS/CS market exists. Part III has argued that OS collaborations should follow something like the five effort rule—the rule that the number of OS companies should be comparatively small.[256] Politicians, on the other hand, may see the fact that "only" one in five companies practices OS as an invitation for continued support. This tendency is especially likely given government's traditional reluctance to dismantle "infant industries" programs that have served their purpose.[257]

C. Tax Policy

Economics textbooks routinely praise the nondistortionary benefits of tax policies that use lump sum taxes and tax breaks to influence behavior. While no government seems to have considered the idea so far, tax policy provides a natural lever for making OS companies less profitable and CS companies more profitable.

---

[251] Singapore, Hong Kong, and Israel. Israel also offers funds for OS start-up companies. *Government Open Source Policies*, *supra* note 236, at 10–11, 15.

[252] Cambodia, China, Czech Republic, Israel, South Africa, South Korea, Japan, Netherlands, Philippines, Thailand, and Vietnam. *Id.* at 5–6, 11–20.

[253] Brazil, Malaysia, Pakistan, Western Australia, and Spain. *Id.* at 4, 12–14, 16, 21.

[254] Brazil. *Id.* at 4.

[255] Despite a very large data set, Professors Lerner and Schankerman were unable to determine whether government subsidies favor OS or CS on net. LERNER & SCHANKERMAN, *supra* note 16, at 199–200.

[256] *See supra* text accompanying notes 135–138.

[257] *E.g.*, William E. Schrank, *Introducing Fisheries Subsidies* 7 & n.15 (FAO Fisheries Technical Paper No. 437, 2003), *available at* http://www.fao.org/DOCREP/006/Y4647E/y4647e00.htm.

This would almost certainly shift the OS/CS ratio toward the five effort rule's 15–20 percent target.

The great advantage of lump sum tax policies is that—unlike purchasing preferences or subsidies—they do not change companies' incentives.[258] Indeed, the only way for companies to evade such taxes is to go out of business. This means that we can confidently expect industry to go on making the same investment decisions as if government had never intervened. For this reason, a well-designed tax policy would eliminate our "crowding out" and "costs to government" objections. Only "fine-tuning"—that is, knowing when the right OS/CS ratio has been reached—remains.

D. Direct Investment in OS

Each of the foregoing policy levers is designed to strengthen CS companies so that increased competition forces OS firms to develop more software. But this seems terribly indirect. If the goal is to develop more OS code, why not purchase it directly? To some extent, government grants do this already.[259] This usually happens informally when grant agencies let individual faculty—most of whom are ideologically predisposed to OS—decide how to license their government-funded software. Furthermore, many American grant agencies—notably NIH—now require grant applicants to provide "dissemination strategies" for sharing their data and discoveries with the wider world.[260] OS licenses provide a particularly simple way to meet these requirements. Indeed, the United Kingdom has taken this logic one step further by adopting interim regulations that make OS licenses the "default position" for government-funded software.[261] Finally, many governments fund projects, institutes, alliances, and consortia whose missions include writing OS software.[262]

Not surprisingly, purchasing still more OS from the private sector aggravates the usual imbalance in mixed OS/CS industries. On the one hand, government contracts make OS companies more profitable. This increases their numbers and

[258] N. GREGORY MANKIW, PRINCIPLES OF ECONOMICS 253 (4th ed. 2007).

[259] For instance, Open Source Victoria received a $50,000 grant in 2003. *Government Open Source Policies*, *supra* note 236, at 21.

[260] *See, e.g.*, *NIH Data Sharing Policy and Implementation Guidance*, NAT'L INSTS. HEALTH, http://www.grants.nih.gov/grants/policy/data_sharing/data_sharing_guidance.htm (last visited Jan. 17, 2012).

[261] The regulations provide: "[I]f no exploitation route is specified for government-funded R&D software outputs, the default position of the government should be 'to adopt an open source software license which complies with the OSI definition (which includes the GPL and Berkeley style licenses) or a United Kingdom-specific analogue of it' [and] 'all government-funded software should be accompanied by appropriate documentation which will assist the exploitation via the open source software license.'" *Government Open Source Policies*, *supra* note 236, at 18.

[262] China, Finland, Japan, South Korea, France, India, Slovakia, Spain, Thailand, Venezuela, and Vietnam. *Id.* at 5, 8–10, 12, 15–20.

makes the ratio of OS-to-CS companies even less favorable than it was before.[263] On the other hand, each dollar that government spends to develop OS code dilutes companies' incentives to invest their own money. This aggravates the cartel effect. Instead of closing the gap in Figure 4, then, direct government investment widens it still further. Despite this, the glass is at least half-full. Cartel effect or not, direct investment increases the amount of OS that society can use and enjoy. Does it really matter whether this investment is made by taxpayers instead of shareholders?

Conservatives will object that this kind of government-funded R&D is bound to cost more than commercial OS. Furthermore, direct support could easily substitute government design choices for market signals.[264] And indeed, poorly designed funding schemes—for example, paying a computer science professor to create her personal vision of "ideal" cell phone software—could easily end in disaster. At the same time, these objections are not really fundamental. The trick will be to design schemes that encourage companies to work cost-effectively on software projects that the market actually wants. This could be done by, for example, inviting commercial OS collaborations to tender competing bids that promise to implement a specific software idea at a guaranteed price. Government would then select whichever software/cost pair promised the most value. Because OS members expect new software to make their complementary goods more desirable, winning bids would usually come in well below cost. In order to make a profit companies would have to propose software that consumers actually wanted.[265]

VII. CONCLUSION

The new commercial OS provides a potentially powerful vehicle for shared software development. At the same time, sharing also creates a de facto cartel that limits investment. This can stop OS output far short of its theoretical potential.[266]

Antitrust doctrine can only do so much to fix this problem. Because OS sharing delivers important benefits, the rule of reason will usually justify significant cartelization. The most judges can do—and it is a great deal—is to make sure that this cartelization does not spread unnecessarily. Judges should be particularly wary of OS collaborations that fall outside (a) operating systems, and (b) the five effort rule. Viral licenses should also receive scrutiny. We have argued

[263] *See supra* text accompanying notes 71–76, 93–98.

[264] Schmidt & Schnitzer, *supra* note 234, at 495 ("[A] government-sponsored research lab does not face the constraints of the market and has much less incentive to focus on customer needs and cost efficiency").

[265] STEPHEN M. MAURER & SUZANNE SCOTCHMER, *Procuring Knowledge*, *in* 15 ADVANCES IN THE STUDY OF ENTREPRENEURSHIP, INNOVATION AND GROWTH 1, 26–27 (Gary Libecap ed., 2004).

[266] *See supra* text accompanying notes 58–107.

that GPL-style licenses should normally be struck down and that even weakly viral, Mozilla-type licenses may not always be necessary.[267]

In the long run, only government intervention can eliminate the cartel effect. Initiatives to reset the OS/CS balance by enacting tax policies that make CS more profitable provide an elegant—if politically unsightly—solution to this problem. Direct government funding of OS projects would also work and is politically more palatable. Here, the principal danger is that poorly designed schemes could replace market signals of consumer need with government's own, top-down vision.

The shock of traditional OS—that people both can and do produce valuable software for non-monetary reasons—has never entirely worn off. For this reason, many observers still see OS as a fragile bloom that must be protected and encouraged. This view is increasingly out of touch: today's OS is commercial, hardheaded, and durable. For this reason, it is time to shift the goal from simply "promoting OS" to getting the OS/CS balance right—and then knowing when to stop. This new viewpoint will make our legal and policy choices harder but also much more interesting.

[267] *See supra* text accompanying notes 145–230.

Table 1: Top 10 Eclipse Contributors[268]

Company (Membership Status)	**Total (Share) of Commits 2001–2010**	**Total (Share) of Commits in 2010**	**Number of employee programmers since 2001**	**Company Profile**
1.IBM (Strategic Member)	3,079,053 (42.6%)	173,171 (16.7%)	388	Hardware, software, and consulting services.
2.Oracle (Strategic Member)	375,810 (5.2%)	84,880 (8.2%)	45	Business management software specializing in information management.
3.Intalio Inc. (Supplier Member)	368,686 (5.1%)	27,824 (2.7%)	10	Business management software and services including Intalio|Designer, an Eclipse-based integrated development[269]
4.Cloudsmith, Inc.	289,534 (4.0%)	76,047 (7.3%)	12	Web Services that help developers find, share, and use OS software[270]
5. Actuate Corp. (Strategic Member)	204,828 (2.8%)	0 (0.0%)	58	Business management and reporting software
6. itemis AG (Strategic Member)	143,073 (2.0%)	49,678 (4.8%)	15	Model driven software development (MDSD) and Eclipse based tool chains.

[268] Data provided by Mike Milinkovic and Wayne Beaton (Eclipse Foundation) and analyzed by Severin Weingarten (Friedrich Schiller University, Jena). Columns 2 and 3 do not include commits by individuals and unknowns. Company profiles are drawn from the Eclipse member pages at ECLIPSE FOUNDATION, http://www.eclipse.org/membership/ (last visited Mar. 13, 2012). Additional company information is drawn from Wikipedia. *See Actuate*, WIKIPEDIA, http://en.wikipedia.org/wiki/Actuate; *IBM*, WIKIPEDIA, http://en.wikipedia.org/wiki/IBM; *Oracle*, WIKIPEDIA, http://en.wikipedia.org/wiki/Oracle_Corporation.

[269] *Company Overview of Intalio, Inc.*, BLOOMBERG BUSINESSWEEK, http://investing.businessweek.com/research/stocks/private/snapshot.asp?privcapId=141317 (last visited Mar. 13, 2012).

[270] *Company Overview of Cloudsmith, Inc.*, BLOOMBERG BUSINESSWEEK, http://investing.businessweek.com/research/stocks/private/snapshot.asp?privcapId=38942939 (last visited June 7, 2012).

7.Borland Software Corp. (Strategic Member)	139,308 (1.9%)	5,895 (0.6%)	11	Mass-market software.
8. Red Hat Inc. (Solutions Member)	117,280 (1.6%)	14,884 (1.4%)	18	Pre-packaged mass-market software.
9.Tasktop (Solutions Member)	116,734 (1.6%)	10,391 (1.0%)	7	Task management solutions for Eclipse.
10. Soyatec (Solutions Member)	125,371 (1.5%)	50,070 (4.8%)	4	Software and Eclipse solutions.