GSPP10-006

Abstract:
Modern open source ("OS") software projects are increasingly funded by commercial firms that expect to earn a profit from their investment. This is usually done by bundling OS software with proprietary goods like cell phones or services like tech support. This article asks how judges and policymakers should manage this emerging business phenomenon. It begins by examining how private companies have adapted traditional OS institutions in a commercial setting. It then analyzes how OS methods change companies' willingness to invest in software. On the one hand, OS cost-sharing often leads to increased output and benefits to consumers. On the other, these benefits tend to be limited. This is because sharing guarantees that no OS company can offer consumers better software than any other OS company. This suppresses incentives to invest much as a formal cartel would. In theory, vigorous competition from non-OS companies can mitigate this effect and dramatically increase OS output. In practice, however, *de facto* cartelization usually makes the OS sector so profitable that relatively few proprietary companies compete. This poses a central challenge to judges and policymakers.

Antitrust law has long recognized that the benefits of R&D sharing frequently justify the accompanying cartel effect. This article argues that most commercial OS collaborations can similarly be organized in ways that satisfy the Rule of Reason. It also identifies two safe harbors where unavoidable cartel effects should normally be tolerated. That said, many OS licenses contain so-called "viral" clauses that require users who would prefer to develop proprietary products to join OS collaborations instead. These clauses aggravate the cartel effect by further reducing the number of companies willing to adopt proprietary business models. Although viral licenses may sometimes be needed to stabilize OS collaborations against free-riding, practical experience suggests that many viral licenses (notably including the GPL) are too broad to survive a Rule of Reason analysis. The article concludes by asking how policymakers can use taxes and government spending to mitigate the cartel effect and/or increase OS software production.

Keywords: Open source; antitrust, monopolization. Sherman Act.
JEL Classifications: K21, L17, O38.

The Penguin and the Cartel: Rethinking Antitrust and Innovation Policy for the Age of Commercial Open Source

By

Stephen M. Maurer
Goldman School of Public Policy
and Berkeley Law School
University of California, Berkeley

August 2010

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 non-commercial, "fun" motives like altruism, hobbyist interest, and the like.[1] By contrast, today's OS projects are mostly commercial. Even if our theories and policy prescriptions were right ten years ago, then, 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 software, no company can offer better 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 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. Section 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. Section III examines how companies make investment decisions in OS, closed source ("CS"), and mixed OS/CS markets. Section 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 proposes two safe harbors in which OS collaborations should be presumed to be pro-competitive. Section V examines the antitrust status of so-called "viral" licenses. It argues that very broad licenses (notably GPL) are unnecessarily restrictive and violate the Sherman Act. Section VI reviews how governments can use taxes, grants, and procurement policy to help OS sharing reach its full potential. Finally, Section 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 narrative. In this telling, OS was a "movement" driven by psychological ("altruism"), political ("ideology"), or post-modern incentives ("the gift economy") that defied conventional economic analysis.[2] The more scholars studied OS, however, the less strange it seemed.

[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 285 (Terrence Hendershott ed., 2006).

[2] 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 (1999).

Indeed, many OS incentives – for example a desire for education and demonstrating one's ability to potential employers – were transparently financial. By 2002 or so, then, economists had come to understand OS almost entirely in terms of familiar motives like education and signaling.[3] At the same time, commercial incentives played a relatively small role and were seldom discussed.[4]

Worldwide OS production rose slowly during the phenomenon's first decade. [Fig. 1] As late as 2002, only 500 OS projects existed.[5] This changed dramatically, however, after software companies began paying their software engineers to join OS collaborations as volunteers.[6] That same year, IBM announced that it would invest a spectacular $1 billion in LINUX.[7] Two years later the total number of OS collaborations had grown fivefold. By 2007 it had doubled again to more than 4,500 projects.[8]

[3] *See, e.g.*, JOSH LERNER & JACQUES TIROLE, SOME SIMPLE ECONOMICS OF OPEN SOURCE, 50 *J. Indus. Economics* 197 (2002); see *also*, MAURER & SCOTCHMER, *supra* note 1.

[4] *Id.*Both sources treat commecial motives as a distinctly subsidiary phenomenon.

[5] These estimate is based on a search engine-driven inventory of on-line OS code depositories. AMIT DESHPANDE & DIRK RIEHLE, *Total Growth of Open Source in* PROCEEDINGS OF THE FOURTH CONFERENCE ON OPEN SOURCE SYSTEMS (2008), http://www.riehle.org/publications/2008/the-total-growth-of-open-source/. The number of OS collaborations has continued to grow exponentially since the paper was published. Dirk Riehle (personal communication).

[6] Survey researchers noticed a secular shift from hobbyist-volunteers to professionals working on company time as early as 2002. *See, e.g.,* STEFANO F. COMINO *et al.* FROM PLANNING TO MATURE: ON THE DETERMINANTS OF OPEN SOURCE TAKE OFF (2005); R. Ghosh *et al.*, FREE/LIBRE AND OPEN SOURCE SOFTWARE: SURVEY AND STUDY (2002).

[7] INTERNATIONAL INSTITUTE OF INFONOMICS, "FREE/LIBRE OPEN SOURCE SOFTWARE: SURVEY AND STUDY PART II" 12 (2002). http://www.berlecon.de/studien/downloads/200207FLOSS_Activities.pdf.

[8] DESHPANDE & RIEHLE, *supra* note 6.

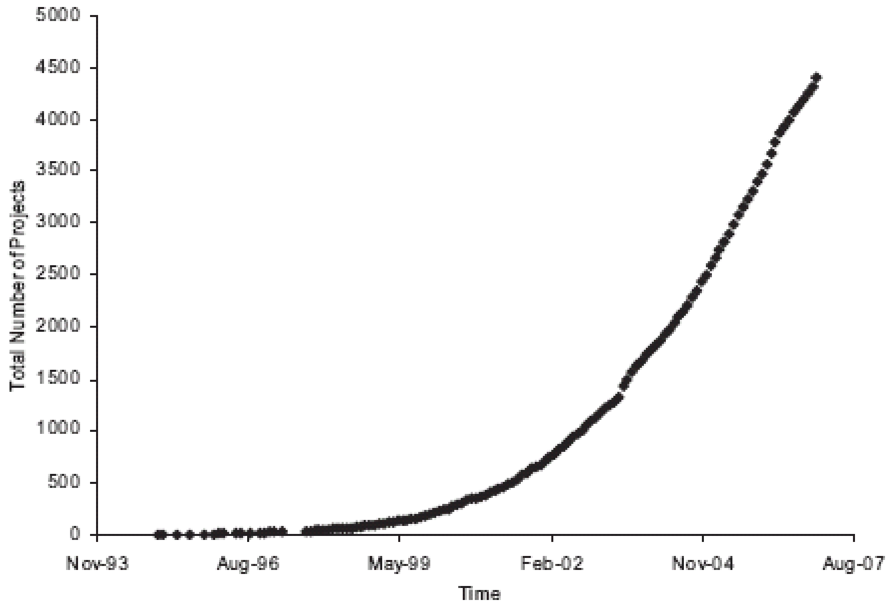

Fig. 1: Number of OS Projects, 1993 – 2007[9]

Although exact figures are hard to come by, observers universally agree that most of the new projects are commercial, *i.e.* funded by for-profit companies that expect a clear dollars-and-cents return on their investment. 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. This complementary good can either be a physical chattel like cell phones, a service like tech support, or closed source ("CS") software.[10]

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. In the New Economy, they encourage employees to join self-governing OS collectives on company time.[11] Furthermore, companies do little or nothing to monitor these activities.

[9] *Id.*

[10] 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 (2009). http://dirkriehle.com/2009/05/01/the-commercial-open-source-business-model/. 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.

[11] Companies' readiness to support paid OS volunteers can be remarkably explicit. For example, the Eclipse Foundation's rules require "Developer"-level corporate sponsors to assign at least eight full-time software developers to the project. "Consumer"-level members who contribute full-time developers similarly pay lower dues. ECLIPSE FOUNDATION, MEMBERSHIP RIGHTS, http://www.eclipse.org/membership/become_a_member/membershipTypes.php.

The Eclipse Foundation. To get a better feel for commercial OS, consider the Eclipse Project. In April 1999 IBM launched a $40 million project to create an "Integrated Development Environment" for professional software developers. Two-and-a-half years later it released Eclipse 1.1.0 as a conventional CS software product.[12] Just one month later, however, it transferred the work to an OS "Eclipse Project" for further development. Legally, the key step was adopting a "Common Public License"[13] so that users could freely use, modify and distribute the code to others.[14] In February 2004 IBM surrendered still more control by moving the Eclipse project to an independent non-profit corporation.[15] At this point, large numbers of outside developers began to join the project.[16] Today, Eclipse has roughly 150 member companies. Roughly one-third of these have donated software to the project since 2001.[17]

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 since 2007.[18] On the other, Eclipse has enjoyed significant market penetration.[19] This has made it the leading tool for developing the software for mobile devices like cell phones, GPS units, and MP3 players[20] as well as enterprise software for data-intensive problems like train scheduling and banking.[21]

---

[12] SEBASTIAN SPAETH *et al*,. ENABLING KNOWLEDGE CREATION THROUGH OUTSIDERS: TOWARDS A PUSH MODEL OF OPEN INNOVATION, __ *International Journal of Technology Management*__ (2010).

[13] http://www.opensource.org/licenses/cpl1.0.php.

[14] 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 FOUNDATION, ECLIPSE PUBLIC LICENSE V.1.0. http://www.eclipse.org/legal/epl-v10.html.

[15] *Id.* The new body is called Eclipse.org. *See also* ECLIPSE FOUNDATION, BYLAWS OF ECLIPSE FOUNDATION, INC., http://www.eclipse.org/org/documents/Eclipse%20BYLAWS%202003_11_10%20Final.pdf

[16] SPAETH, *supra* note 12.

[17] *See* ECLIPSE FOUNDATION, EXPLORE THE ECLIPSE MEMBERSHIP. http://www.eclipse.org/membership/exploreMembership.php; ECLIPSE FOUNDATION, ECLIPSE DASHBOARD, http://dash.eclipse.org/dash/commits/web-app/summary.cgi. At least 27 non-member companies have also contributed software.

[18] Mike Milinkovich, personal communication.

[19] 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 (Red Hat), Xtensa Xplorer IDE (Tensilica), Time Storm IDE (TimeSys), and Workbench (Wind River). ROBERT DAY, ECLIPSE HERALDS TRUE INTEROPERABILITY, *Electronic Engineering Times* (April 3, 2006); DARRYL K. TAFT, ECLIPSING CHALLENGES: GROUP'S DIRECTOR TAKES ON BUSINESS, TECHNOLOGY," *eWeek*, Nov. 15, 2004.

[20] TAFT, *supra* note 19.

Why do companies participate? The "business advantages," according to Eclipse, depend on cost-sharing.[22] At first blush, this rationale may sound homely to readers accustomed to claims that OS outperforms CS methods in productivity,[23] finding and fixing bugs,[24] or eliciting clever product ideas from users.[25] But consider the math: OS advocates would be overjoyed if they could cut costs or increase quality by, say, twenty percent.[26] Cost sharing, on the other hand, cuts per-firm costs *fifty* percent the first time that two firms collaborate and goes on generating additional (if smaller) savings for each firm that joins thereafter.[27] On any reasonable assumption, then, we expect sharing to dominate other effects.

What kinds of companies join Eclipse? Table 1 profiles the top fifteen companies that have donated software to the Project since 2001.[28] 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 (26%) of all programmers and two-fifths (39%) of all software deposits since 2001.[29] Even so,

---

[21] ECLIPSE FOUNDATION, RICH CLIENT PLATFORM APPLICATIONS. http://www.eclipse.org/community/rcp.php.

[22] More precisely, Eclipse says that it allows "individuals and companies … to collaborate on projects that would be difficult to achieve on their own." ECLIPSE FOUNDATION, ECLIPSE PUBLIC LICENSE (EPL) FREQUENTLY ASKED QUESTIONS. http://www.eclipse.org/legal/eplfaq.php. Eclipse adds that its model has the added, "technical advantage" of "turning users into potential co-developers." *Id.*

[23] RAYMOND, *supra* note 2.

[24] *Id.*

[25] ERIC VON HIPPEL, DEMOCRATIZING INNOVATION (2005).

[26] For a sense of the still-embryonic research on the comparative efficiency of OS and CS methods, *see* STEFAN KOCH, PROFILING AN OPEN SOURCE ECOLOGY AND ITS PROGRAMMERS, 14 *Electronic Markets* 77 (2004) and JAI ASUNDI, *Need for Effort Estimation Models for Open Source Software Projects in* FIFTH (ACM) WORKSHOP ON OPEN SOURCE SOFTWARE ENGINEERING (2005).

[27] 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 slows, but does not eliminate, OS sharing. *See, e.g.*, JUSTIN PAPPAS JOHNSON, OPEN SOURCE SOFTWARE: PRIVATE PROVISION OF A PUBLIC GOOD, 24 *J. Econ. & Indus'l Strategy* 637 (2002).

[28] 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.

[29] Data obtained from the Eclipse collaboration's statistical "dashboard" tool on May 3, 2010. ECLIPSE FOUNDATION, ECLIPSE DASHBOARD, http://dash.eclipse.org/dash/commits/web-app/summary.cgi. These data appear to be reasonably complete. Only two percent (147,826) of all commits are attributed to "unknown" sources.

IBM's glass is more than half full: After all, sixty percent of all deposits were paid for by someone else.

Other Dominant Companies. The next three companies in Table 1 – Oracle, Intalio, and Actuate – contributed an additional ten percent of all software deposits. Together with IBM, this shows that about half of the total Eclipse effort since 2001 has come from four large players. To this point, the picture is not too different from a conventional joint venture.

Small Companies. Eclipse attributes the remaining fifty percent of all software donations to 178 small companies and hundreds of individuals. While the precise statistics are uncertain, small businesses probably account for about forty percent of all donations since 2001.[30] At least thirty-six of these companies are substantial contributors, *i.e.* made more than 10,000 software deposits over the life of the project.[31]

Traditional OS Participants. The remaining Eclipse deposits come from sources that resemble traditional, non-commercial OS communities. These include non-profits, academic organizations, and hundreds of individuals.[32] While these participants account for relatively few deposits – perhaps ten percent or so – this probably understates 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.[33]

Sharing is not the whole story, of course – Companies must also find a way to recover their investments. Table 1 profiles the top fifteen corporate software donors since 2001. Strikingly, all fifteen companies earn their living by making and selling CS products like

---

[30] Eclipse's records show that small companies account for twenty percent of all software deposits while individuals account for thirty percent. However, the number of individual contributions is almost certainly overstated. This is 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." ECLIPSE FOUNDATION, DASH PROJECTS COMMITS EXPLORER. http://wiki.eclipse.org/Dash_Project/Commits_Explorer. It is hard to estimate the size of this effect. However, it is worth noting that just ten percent of programmers are responsible for three-quarters (74%) of all deposits by individuals since 2001. Here, the level of required effort is so high – at least 10,000 commits – that some kind of business support seems necessary. This provides the basis for our very rough estimate in text that small businesses contribute roughly forty percent of all software deposits compared to just ten percent for individuals.

[31] *Id.* Sixty-nine companies contributed more than 1,000 separate deposits over the life of the project.

[32] Academic and non-profits that have donated software over the life of the project include INRIA, Centrum voor Winkunden Informatica, Social Physics, Insitut fur Software, and Ecole Polytechnique de Montreal.

[33] VON HIPPEL, *supra* note 25.

hardware, software, and/or IT consulting services. Directly or indirectly, these sales pay for Eclipse.

Making Sharing Work: Collaboration Architecture. However reasonable, 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, for example, signing joint venture agreements. Why should OS work any better? One partial answer is that ordinary joint ventures are deeply flawed.[34] 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.

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 a Board of Directors which includes (a) representatives of 15 large corporate sponsors, and (b) six elected members representing the committer and add-on communities.[35] The Board sets the Foundation's operating budget, decides which new software projects to develop, and can also amend the Foundation's OS license.[36] Because Board membership is diverse, no single interest group – including IBM and other large players – has nearly enough votes to hijack the collaboration.[37]

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. Instead, volunteers are encouraged to discard

---

[34] For the 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). http://www.law.harvard.edu/programs/olin_center/corporate_governance/papers/03.majwski-williamson.incomplete-contracting.pdf; OLGA FILIPENKO, OPTIMALITY OF JOINT VENTURES AND STRATEGIC ALLIANCES (2001). http://www.stern.nyu.edu/fin/pdfs/seminars/013f-filipenko_phd.pdf.

[35] The current Board consists of IBM, Red Hat, Nokia, SAP, Brox, Actuate, Sopera, Innoopract, OBEO, Oracle, CA Inc., itemis AG, Cloudsmith, Sonatype, Genuitec, and six individuals elected by the committer and add-on provider communities. Eclipse Foundation, Eclipse Foundation Board of Directors. http://www.eclipse.org/org/foundation/directors.php.

[36] The Board also hires and fires the Foundation's director, sets membership fees, terminates and reinstates members, and sets IP and antitrust policy. ECLIPSE FOUNDATION, TYPES OF MEMBERSHIP. http://www.eclipse.org/membership/become_a_member/membershipTypes.php

[37] 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. ECLIPSE FOUNDATION, MEMBERSHIP RIGHTS. http://www.eclipse.org/membership/become_a_member/memberRights.php.

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."[38]

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. Better to rely on collaboration insiders – *i.e.* committers and ordinary volunteers – who are actively involved in the project. 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 fails to deliver value. These "efficiency wages" create a strong incentive to please employers.[39] They also work indirectly by encouraging lower-level insiders to work hard so that they, too, can earn efficiency wages in the future.
>
> *Collaboration Assets.* Insiders can usually extract rents from the collaboration over and above the value of their labor. 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,[40] and (c) money and in-kind contributions from corporate sponsors to the collaboration as a whole.[41] 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 knowing. But why should the other insiders permit this? After all, their income depends on the

[38]ECLIPSE FOUNDATION, ECLIPSE DEVELOPMENT PROCESS. http://www.eclipse.org/projects/dev_process/development_process.php#4_1_Committers

[39] For a description of efficiency wages *see, e.g.*, WALTER NICHOLSON, MICROECONOMIC THEORY: BASIC PRINCIPLES AND EXTENSIONS 430 (6th ed. 1995). Nicholson notes that the empirical evidence for efficiency wages is thin. *Id.*

[40] 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 3. Here, the prospect of a future job acts as a prize to induce current effort.

[41] Typical examples include paying for workshops and conferences, purchasing hardware, and making previously-written CS code available under GPL. JOEL WEST AND SIOBHAN O'MAHONEY, CONTRASTING COMMUNITY BUILDING IN SPONSORED AND COMMUNITY FOUNDED OPEN SOURCE PROJECTS, IEEE PROCEEDINGS OF THE 38TH ANNUAL HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES (2005).

> collaboration's reputation for delivering value. This gives them a direct stake in seeing that all sponsors receive value – and to block outcomes that unfairly deprive sponsors of value.

Are any of these arguments correct? The jury is still out.[42] Nevertheless, companies have clearly found them sufficiently persuasive to justify ambitious experiments like Eclipse. Until we learn differently, then, it is only prudent to assume that OS sharing is efficient. 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 their software development costs. On the other, shared code strengthens their competitors.[43] This can reduce profits to the point where OS no longer makes sense. Finally, sharing ensures that each OS company offers consumers the same 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 section explores OS economics in detail. We begin by asking how companies in a traditional all-CS industry decide how much to invest in software. We then explore how sharing changes this analysis for an all-OS industry. Finally, we turn to mixed markets in which OS and CS companies compete with one another. We will see that strong CS competition can force OS collaborations to produce far more software than a pure-OS industry would. However, we will also see that this condition seldom occurs in practice. Instead, market forces tend to damp CS competition so that OS collaborations generate much less software than they theoretically could. This gap between OS software's potential and its actual performance poses a central challenge to judges and policymakers.

How CS Firms Invest. We start by asking how companies in a traditional, all-CS industry decide whether and 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

[42] 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 are tempted to delay effort in hopes that someone else will do the work. *See, e.g.,* JUSTIN PAPPAS JOHNSON, *supra* note 27. Alternatively, small companies may sometimes withhold effort knowing that their contributions do not matter much in any case. We expect the result in both cases to be (a) that large companies shoulder more than their "fair share" of effort, and (b) that total effort across the collaboration is smaller than it otherwise would be.

[43] 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 FOUNDATON, *supra* note 38.

manufacture and sell a product that contains the software in Year Two.[44] This second product can be a chattel (*e.g.* a cell phone), a service (*e.g.* tech support), or CS software.[45] Readers should note that this scenario includes two decisions: (a) How much to invest in software in Year One and (b) How many goods and services to place on the market in Year Two. Each of these implies a separate and distinct forum for competition between companies.

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. We begin by analyzing the company's Year Two problem. The solution is routinely taught in college economics courses[46] and is more or less what you would expect: CS output rises (and profits fall) as competition increases in the second product. This article will refer to this dynamic as "Year Two" or "quantity competition" in what follows.[47]

Solving the Year Two problem 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, *i.e.* how much software is written in Year One. Detailed calculation[48] 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" or "quality competition."

---

[44] For a more formal version of the analysis presented in this section, *see* STEPHEN M. MAURER & SEBASTIAN VON ENGELHARDT, THE NEW (COMMERCIAL) OPEN SOURCE: DOES IT REALLY IMPROVE WELFARE? *Goldman School of Public Policy Working Paper No.* GSPP10-001 (2010) 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.* CLOSED SOURCE BUSINESS MODELS, Jena Economic Research Paper 2010-034 (2010). http://pubdb.wiwi.uni-jena.de/pdf/wp_2010_034.pdf. Profs. Llanes and de Elejalde use a similar analysis to investigate how firms choose between OS and CS business models. GASTON LLANES AND RAMIRO DE ELEJALDE INDUSTRY EQUILIBRIUM WITH OPEN SOURCE AND PROPRIETARY FIRMS," *Harvard Business School Working Paper*, No. 09-149 (2009). http://ssrn.com/abstract=1425549.

[45] 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 to. The case is more subtle for software which consumers can frequently copy at zero 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.

[46] 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 *x*'s for every company add up to a self-consistent, industry-wide *X*. *See.*, *e.g.*, WALTER NICHOLSON, MICROECONOMIC THEORY: BASIC PRINCIPLES AND EXTENSIONS (6th ed. 1995).

[47] MAURER & VON ENGELHARDT, *supra* note 45.

[48] *Id.*

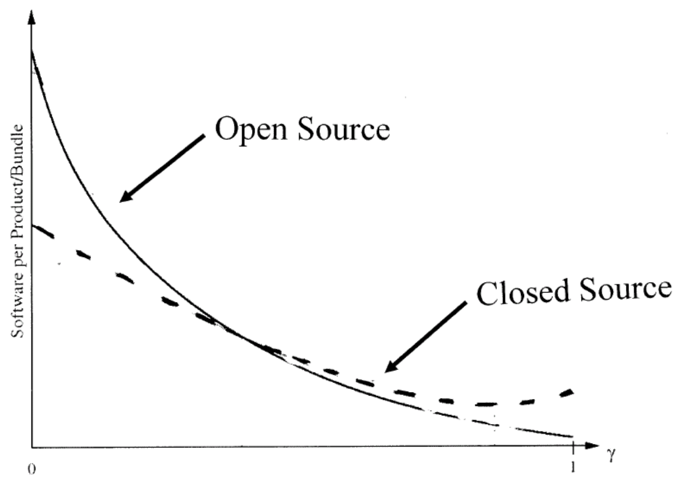

Fig. 2: OS and CS Investment *vs.* Competition in Year Two Product[49]

Fig. 2 summarizes these arguments. Here, the horizontal axis denotes how similar companies' Year Two products are to one another and increases from left (completely different) to right (completely interchangeable). Readers should note, however, that competition depends on product similarity. For this reason, we will usually think of the horizontal axis as an index of Year Two competition.[50] This means, *inter alia*, that the dashed line shows how much software a typical CS company creates as a function of competition. Production is highest on the left-hand side where each product is unique so that competition is minimal. This makes sense since each company is effectively a monopolist and knows that it can recover a large fraction of whatever benefits its software confers on consumers. Conversely, CS software production falls as competition increases.

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 Arm's Races, these investments cancel leaving each company no better off – and considerably poorer – than it was before. This process is painfully familiar in Silicon Valley where CEOs frequently complain about razor-thin margins for "commoditized" (*i.e.* generic) software.[51]

[49] Adapted from MAURER & VON ENGELHARDT, *supra* note 45.

[50] 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.

[51] *See, e.g,.,* IAN MURDOCK, *Open Source and the Commoditization of Software* in OPEN SOURCES 2.0 (Chris Dibona *et al.* eds., 2008). http://commons.oreilly.com/wiki/index.php/Open_Sources_2.0/Open_Source:_Competition_and_Evolution/Open_Source_and_the_Commoditization_of_Software

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. These temporary monopolies are far more lucrative than creating new versions of existing products.[52] But what should the rest of us think? Outside Silicon Valley, most products are already commoditized – *i.e.* produced by multiple competing companies. But in that case, our usual antitrust intuition suggests that commoditization is good for society. And indeed, detailed economic models of CS production confirm that commoditization usually helps consumers more than it hurts industry.[53] While it is possible to invent scenarios in which software investment delivers negative value overall, these cases are almost certainly rare.[54]

How OS Firms Invest. Now consider how a company in an all-OS industry invests. Start by noticing that once the software exists, the company's Year Two/quantity competition choices are the same as a CS company's. This means that we need only consider the company's Year One strategy. Here, sharing introduces two important changes. On the one hand, it almost always reduces per-firm costs 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 attactive.

To see how these effects interact, consider the solid line in Fig. 1. OS investments are a no-brainer on the left hand side where competition is weak. This is because companies can split costs without threatening each others' revenues. What could be better?[55] 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 righthand side of Fig. 1.

Even so there is a surprise. Unlike the CS case, OS output never encounters a commoditization crisis and never stops falling. How can this be? The reason is sharing.

[52] 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).

[53] MAURER & VON ENGELHARDT, *supra* note 45.

[54] 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.

[55] Actually, one *can* imagine something better. As Prof. Schmidke points out, many goods are complements, *i.e.,* improvements to one product increases demand for the other. In this situation, firms that invest in R&D confer benefits on both themselves and others. Since only the former are compensated, this leads to under-investment. OS sharing gets around this externality by creating a channel through which companies can subsidize each others' products. RICHARD SCHMIDKE, PRIVATE PROVISION OF A COMPLEMENTARY PUBLIC GOOD. CESifo Working Paper 1756 (2006).

By definition, no company in an all-OS industry can offer consumers more software than any other company. This suppresses quality competition much as a formal cartel would.[56] While OS companies invest in software, then, they stop writing sooner than they would if quality competition existed.[57] This leads to the usual monopoly result – An under-supplied product.[58]

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 which suppresses OS companies' incentives to invest in the first place. We therefore expect OS collaborations facing stiff Year Two competition to produce less software than CS companies. This can still be good for society since each OS member would otherwise have to write its own duplicative program. On the other hand, OS companies would have written much more code absent the cartel effect. The glass is only half-full and we expect both OS-enthusiasts and policymakers to be disappointed.

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. Can CS companies deliver the missing quality competition? Simple calculations are encouraging. For example, von Engelhardt and Maurer find that mixed OS/CS industries deliver maximum benefits when fifteen to twenty percent of all companies practice OS.[59] [Fig. 3]

---

[56] Ironically, companies which share software produce even 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. This reduces their incentive to produce code in the first place. MAURER & ENGELHARDT, *supra* note 45.

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 45; RAMON CASADESUS-MASANELL & GASTON LLANES, *Mixed Source*, NET Institute Working Paper 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 decision into the Maurer and von Engelhardt model likewise suppresses cartel effects. While mathematically convenient, the Llanes *et al.* decision fails to capture the much longer lead times for creating software compared to the manufacture of physical goods.

[57] 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 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."

[58] In retrospect, this should not surprise us. Scholars and judges have long worried that conventional CS sharing (*i.e.* joint ventures) might suppress competition in innovation. *Addamax Corp. v. Open Software Foundation, Inc*,. 888 F. Supp. 274 (1995) (joint venture could potentially cartelize software production by participants) *aff'd.* 152 F.3d 48, 52 (1st Cir. 1998). This paper argues that OS-sharing entails similar drawbacks.

[59] Careful readers will notice that OS output in Fig. 3 actually peaks when OS companies constitute about ten percent of the market. Social benefits would also peak at this point if they depended solely on how

Reassuringly, this percentage is more or less identical to the rule of thumb that the Department of Justice usually invokes when measuring cartelization threats in, for example, merger guidelines.[60]

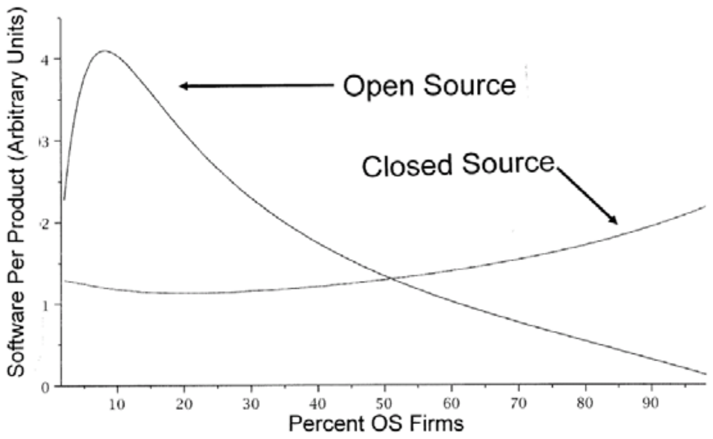

Fig. 3: Software Production *vs.* Percent of Industry Members Adopting OS Methods.[61]

Finally, Fig. 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 Fig. 3 with radically increased sharing. Convincing examples of OS/OS competition can currently be found in several commercial markets including Eclipse *vs.* NetBeans (OS Java tools), Apache Harmony *vs.* OpenJDK (Java runtimes), Pentaho *vs.* Jaspersoft (business reporting software), and Jetty *vs.* Tomcat (web servers). The picture for Content Management Systems – in which OS collaborations Plone, Drupal, PostNuke, and Joomla all compete with one another – is even more encouraging. The main drawback is that we would expect some or all of these projects to fall by the wayside as dominant standards emerge. However desirable, OS/OS competition may not be stable.

Market Imperfection No. 1: Lock-In. We have seen that the benefits of sharing are largest in mixed markets where CS companies outnumber their OS competitors by roughly four to one. Do market force guarantee this result? The answer is almost certainly "no." One

much OS software is produced. In fact, however, CS software also generates benefits by delivering products to 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 fifteen to twenty percent of all companies pursue OS business plans. MAURER AND VON ENGELHARDT, *supra*, n. 45.

[60] Merger review is triggered at a Herfindahl index of 1800. US DEPARTMENT OF JUSTICE, HORIZONTAL MERGER GUIDELINES (rev. 1997). This overlaps the fifteen to twenty percent window calculated by MAURER & VON ENGELHARDT, *supra* note 45.

[61] MAURER & VON ENGELHARDT, *supra* note 45.

reason is that infant industries will often be born in an All-OS or All-CS state.[62] Such industries can become "locked in" so that mixed OS/CS markets never emerge. To see this, consider an infant industry that consists entirely of CS companies. Then OS companies will only enter the market if they expect to earn a profit. However, detailed calculation shows that OS companies cannot do this where quality competition is strong. For this reason, we expect many All-CS markets to be locked in indefinitely.[63] In a few cases, incumbents can even block entrants by deliberately selecting CS where OS sharing would facilitate entry. Like predatory pricing, this strategy requires companies to choose CS even though OS sharing would yield higher profits in the short run.[64]

Lock-in can also occur in All-OS markets. This happens when Year Two competition is so weak that would-be CS entrants cannot earn a profit. Here too, we expect market forces to block more desirable, mixed-states in which OS and CS companies compete with one another.

Market Imperfection No. 2: Too Many OS Firms. Despite lock-in, mixed OS/CS markets will probably emerge in most cases. Even here, however, the OS/CS mix will usually be far from ideal. 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. This implies that most companies will choose OS over CS. Fig. 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 sixty to seventy percent although each company's market share is small. 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.[65]

[62] 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.

[63] MAURER & VON ENGELHARDT, *supra* note 45. The reason for this result is apparent from Fig. 2, which shows that OS collaborations are maximally profitable where quantity competition is weak.

[64] *Id.*

[65] The point is nicely illustrated by the hundreds of small companies that participate in commercial OS collaborations like Eclipse and PLONE. *See, e.g.* ECLIPSE FOUNDATION, EXPLORE THE ECLIPSE MEMBERSHIP. http://www.eclipse.org/membership/exploreMembership.php; PLONE.NET, PLONE PROVIDERS. http://plone.net/providers/.

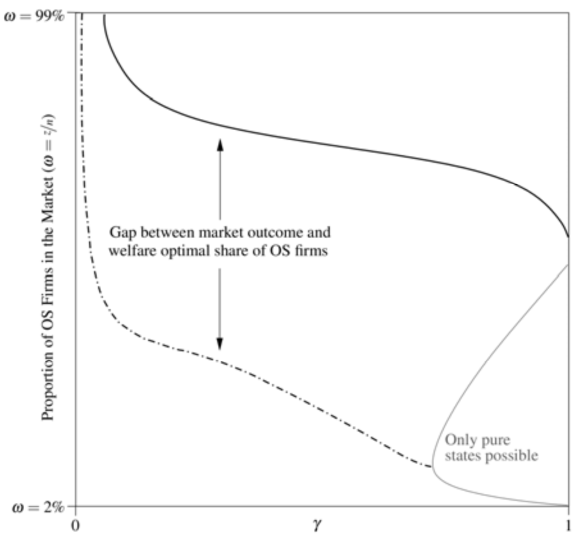

Fig. 4: Ideal (Dashed) and Predicted (Solid) Percentage of Companies Practicing OS Business Models as a Function of Competition.[66]

This systematic over-supply of small OS firms has profound implications. We have seen that policymakers and judges would like OS collaborations to occupy the left-hand side of Fig. 3 where OS sharing delivers much larger software packages than any CS company could hope to build on its own. Because of the cartel effect, however, we expect real OS collaborations to fall on the right-hand side and produce less software than their CS competitors.[67] This anemic result is still better than having each OS company write its own duplicative software.[68] Nevertheless, the situation is far from ideal. Sections IV through VI will evaluate judges' and policymakers' options for closing this gap.

Beyond Simple Models. So far, we have analyzed OS output and the cartel effect using deliberately generic assumptions. This suggests that our insights will apply to most real markets. Nevertheless, details matter. We close this section by asking how more complicated – but still plausible – assumptions could change our understanding of the OS/CS problem:

> *Inefficient Sharing.* We have assumed that OS sharing is efficient. Though plausible, this remains to be seen. Less efficient sharing would (a) reduce the benefits of OS while (b) leaving the cartel effect as large as before. This suggests that udges and policymakers should scrutinize OS more closely where sharing is imperfect.

[66] MAURER & VON ENGELHARDT, *supra* note 45.

[67] 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 2.

[68] Economists usually refer to the wasteful duplication associated with conventional CS production as "business stealing" or "me-too products." *See, e.g.,* ERIC VON HIPPEL & JOACHIM HENKEL, WELFARE IMPLICATIONS OF USER INNOVATION, 30 *J. Techn. Transfer* 73 (2004).

Idiosyncratic Consumers. We have assumed that products compete on both quantity and quality.[69] However, software consumers sometimes show strong loyalties that lead them to ignore large differences in price and/or quality.[70] Here, quality competition will likely be weak in any case.[71] In these circumstances, policymakers and judges could reasonably conclude that any additional damage from cartel effects is not worth worrying about.

Multiple Technologies. This section 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.[72]

Product Differentiation. We have assumed that OS sharing reduces the wasteful duplication inherent in having each company write its own software. 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.

Limited Entry. We have made the conventional assumption that new companies can and will enter the market until incumbents' profits fall to zero. 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

[69] Formally, the assumption leads to models that feature classically linear demand curves. MAURER AND VON ENGELDHARDT, *supra* note 41.

[70] Brand-loyalty can be represented using so-called Hotelling demand models. LLANES AND DE ELEJALDE, *supra* note 41.

[71] Detailed calculations by Drs. Llanes and De Elejalde have confirmed this intuition for a model containing two technology sectors. *Id.*

[72] This is indeed what happens in Drs. Llanes and De Elejalde's analysis when they split their technology sector into an 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 across companies so that each company takes responsibility for whichever project(s) offer the most value to its business. As Prof. 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*, Copenhagen Business School/Aalborg University DRUID Working Papers 06-25 (2006). http://www3.druid.dk/wp/20060025.pdf. Judges and policymakers should be expecially tolerant of cartel effects in these circumstances.

> to make OS companies systematically more profitable and therefore more numerous than CS companies. [73]

Based on this survey, 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 One of the Sherman Act reaches "Every contract, combination … or conspiracy, in restraint of trade …"[74] Under the Rule of Reason, judges and policymakers must intervene whenever an agreement's "anticompetitive effects on trade outweigh its pro-competitive effects."[75] In theory, OS cost sharing eliminates wasteful duplication and lets companies write far more software than they could before. In practice, however, we expect the cartel effect to suppress the second benefit. Does this diminished OS still offer enough benefits to satisfy the Rule of Reason?

This section reviews the antitrust rules for CS cost sharing (*i.e.* 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. We also suggest two "safe harbor" exceptions in which OS collaborations should be presumed valid. The section concludes with a short discussion of monopolization issues under Section Two of the Sherman Act.[76]

Existing Joint Venture Law. There is still surprisingly little case law and commentary analyzing OS licenses under the Sherman Act.[77] Fortunately, the antitrust treatment of CS

---

[73] LLANES & DEELEJALDE, *supra* note 41; RAMON CASADESUS-MASANELL AND GASTON LLANES, *supra* note 50.

[74] 15 USC §1. 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 (1939) (inferring conspiracy from multiple bilateral agreements where parties agreed "knowing that concerted action was contemplated and invited.")

[75] *National Collegiate Athletic Ass'n v. Board of Regents*, 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 justified.")

[76] 15 USC §2.

[77] *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 prices low forever" and cannot "lead to monopoly prices in the future.")

sharing (*i.e.* joint R&D ventures) has been extensively discussed.[78] For most of the 20th Century, antitrust authorities evaluated joint R&D ventures like any other cartel. 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 Prof. 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 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."[79]

This new appreciation for sharing was nevertheless tempered by fears that cartelization might limit companies' R&D investment. 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 "…that the decrease in competitive pressure to innovate will outweigh whatever economies of scale or other efficiencies are likely to be generated by the collaboration."[80] Tests of this criterion included asking (a) whether the agreement "eliminates innovations that would have been cost-justified in a competitive market,"[81] (b) whether the parties would "have developed the product more quickly alone,"[82] and (c) "[w]hat the parents would have done but for the joint venture."[83] More concretely, Congress[84] and the Department of Justice[85] also created a new safe harbor for joint R&D ventures. 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 rule for CS sharing.

---

[78] The US Department of Justice challenged just one joint venture on Sherman Act grounds during the entire 20th Century. ROBERT PITOVSKY, ANTITRUST AND INTELLECTUAL PROPERTY: UNRESOLVED ISSUES AT THE HEART OF THE NEW ECONOMY. 16 *Berkeley Tech. L.J.* 535, 544-545, *citing U.S.. v. Automobile Mfgrs. Assn.*, 307 F.Supp. 617, 621, *affd. sub nom. City of New York v. United States,* 397 US 248 (1970).

[79] EDWARD CORREIRA, JOINT VENTURES: ISSUES IN ENFORCEMENT POLICY, 66 *Antitrust L. J.* 737 (1998).

[80] *Id.*

[81] 13 HERBERT HOVENKAMP ANTITRUST LAW § 2115. *See also, Northrop Corp. v. McDonnell-Douglas Corp.* (9th Cir. 1982) 705 F.2d 1030 (joint venture to invent military aircraft that neither partner could develop separately was not *per se* violation).

[82] HOVENKAMP *supra* note 82.

[83] CORREIA, *supra* note 75.

[84] National Cooperative Research and Production Act, 15 USC §§ 4301-4305.

[85] Department of Justice, "Antitrust Enforcement Guidelines for International Operations" 4 Trade Reg. Rep. (CCH) ¶ 13,109. Hereinafter "1988 Guidelines."

Instead, it merely repeated the Department of Justice's usual rule of thumb for evaluating market concentration in merger cases.[86]

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. This underlying attitude was now reversed by a new intuition that "…the optimal amount of innovation can ordinarily be achieved through … cooperative innovation ventures."[87]

A Rule of Reason Approach to OS Collaborations. We have seen that OS's chief benefit is cost-sharing and its main vice is cartelization. Furthermore, these effects cannot be disentangled: Firms will 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. Section 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 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:

> *Limited Absorptive Capacity.* All projects eventually reach the point where further investment offers negligible consumer benefits. The question is whether cartelization chokes off investment before or after this happens. In general, courts should be more willing to tolerate cartelization where the expected returns from software investment are kinked, *i.e.* fall off sharply beyond some well-defined threshold.
>
> *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.

[86] Prof. 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 75.

[87] CORREIA, *supra* note 75 at n. 82. This expansive proposition is said to be founded on economics research suggesting that even a monopoly "will not under-invest in research." *Id.*

> *Pathological Competition.* We noted in Section III that it is possible to construct scenarios in which quality competion 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 whether these conditions apply is inevitably 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").

Operating Systems. Operating systems provide the clearest candidate for a software product in which the downsides of cartelization can be safely ignored. First, their effect is mediated through applications programs. This means that they have only an indirect influence on the look, feel, and functionalities that consumers actually receive.[88] More speculatively, everyday experience suggests that software has reached the point where even large increases in operating system size produce only incremental improvements. This suggests that the technology is already deep into diminishing returns.

Software engineers frequently say that "operating systems should be open and applications programs should be closed."[89] Apart from the empirical observation that most OS collaborations develop operating systems, however, the policy rationale for such a rule has never been clear. Our analysis validates the first ("operating systems") part of this statement. We now turn to the second part, *i.e.* whether "applications" programs should normally be closed.

Killer Apps. Section III's arguments were predicated on a tacit assumption that program quality depends almost entirely on the amount of programming effort invested. However, not all software derives its value from effort or – in the industry's derisive phrase – "is sold by the pound."[90] Instead, many applications programs derive most of their earning power from cleverness. Here, the most extreme case consists of so-called "Killer App" products that discover previously overlooked needs or invent new functionalities. 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.[91] Part of the reason is practical: Since new

[88] Prof. Ed Lazowska, personal communication.

[89] I am indebted to the late Prof. Richard Newton for this observation.

[90] Mike Milinkovich, personal communication.

[91] KRZYSZETOF KLINCEWICZ, INNOVATIVENESS OF OPEN SOURCE PROJECTS (2005). http://opensource.mit.edu/papers/klincewicz.pdf.

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. Illegality apart, the prospect of an OS cartel offering small-but-predictable rewards does not appeal to them.[92]

Commoditized Apps. Over time, even the most unique products are imitated and become generic. For this reason, most software derives substantial value from *both* ideas *and* raw programming effort. The problem, as we have noted, is that the payoffs from making clever improvements to existing software genres are relatively small. In this case, companies could reasonably decide that it was better 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 Windows product currently faces four OS competitors.[93] On the one hand, this five-way competition surely minimizes cartelization risk for the immediate future. On the other, many of these OS collaborations are overwhelmingly likely to merge or drop by the wayside. This will inevitably strengthen the cartel effect over time.

Proposed Safe Harbors. The Five Effort rule is strikingly consistent with our argument that society receives maximal value when roughly 15 - 20% of all software companies practice OS sharing.[94] This is no coincidence. Section 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 used as an OS safe harbor. The reason is that our arguments do not depend on effort *per se*, but only the number of OS companies which participate in the collaboration. Furthermore, Section III has argued that the amount of effort that OS companies invest can vary widely depending on how much competition they face. For this reason, an OS safe harbor should probably be based on how many companies have joined the collaboration. This revised Five Effort rule would then create a safe harbor for any OS collaboration that included less than twenty percent of all current industry members.

---

[92] Hardware and service providers are even less likely to want this trade given Killer Apps' historic importance in persuading consumers to upgrade computer systems.

[93] Sun's Star Office, IBM's Open Office, Gnome, and the K Desktop Environment.

[94] *See* Fig. 4, *supra*.

It also makes sense to create a second safe harbor for OS collaborations that develop operating systems. We have seen that such collaborations usually derive large benefits from sharing but present only minimal cartelization risks. Here, the main practical difficulty is the fuzzy dividing line between operating systems and applications programs. This could encourage OS collaborations trying to develop applications programs to escape judicial scrutiny by relabeling their projects as "operating systems." That said, this danger seems manageable by limiting the safe harbor to programs that are (a) typically found in today's operating systems, or (b) have become so generic that their inclusion in an operating system is no longer surprising.[95]

Section Two Claims? So far we have concentrated on the idea that OS might violate Section One of the Sherman Act. This is reasonable since a Section Two claim would require proof that violators already possessed a substantial share of the market.[96] We have seen that OS companies hardly ever meet this test.

That said, Section Two issues could still arise on the CS side. Here, the most likely issue is lock-in. We have seen that incumbents can sometimes block entry by new competitors by adopting CS. That said, a Section Two claim seems unlikely. The reason lies in the well-settled case law holding that Section Two does not create a duty to protect competitors.[97] 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,[98] such claims would almost certainly fail.

V. The Sherman Act (B): Viral Licenses

We have argued that – the cartel effect notwithstanding – OS sharing is often justified under the Rule of Reason. At this point, judges must accept some suppression of output. Further attempts to mitigate the cartel effect, if they come at all, will have to come from government.[99] 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

[95] 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. *U.S. v. Microsoft Corp.*, 253 F.3d 34, 84 (D.C. Cir. 2001).

[96] *Spectrum Sports, Inc. v. McQuillan*, 506 U.S. 447, 459 (1993) (monopolization claim required showing of dangerous probability of success predicated on defendant's existing market power).

[97] *Verizon Comms., Inc. v. Law Offices of Curtis V. Trinko,* 540 U.S. 398 (2004).

[98] For a recent review, *see* ROBERT PITOFSKY ET AL., THE ESSENTIAL FACILITIES DOCTRINE UNDER US ANTITRUST LAW, 70 *Antitrust L.J.* 443 (2002).

[99] *See* Section VI, *infra.*

unavoidable. Here, the most urgent issues involve so-called “viral” licenses[100] that require consumers who use OS software to license any improvements or extensions on similar terms.

This section 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 many weaker licenses (*e.g.* Mozilla) should likewise be viewed with suspicion. We close by identifying circumstances in which companies are most likely to adopt viral licenses as a deliberate cartelization strategy.

Antitrust Law and Restrictive Licenses. We have argued that many and perhaps most OS collaborations can be implemented in ways that satisfy the Rule of Reason. However, this does not immunize every “naked restraint” (more precisely: “unnecessary output limiting appendage”) that OS members decide to include in their licenses.[101] Indeed, such terms are *per se* illegal.[102] 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 is therefore “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.[103] On the other hand, judges must still conduct a Rule of Reason inquiry to see whether the benefits of OS sharing could have been achieved by adopting some less restrictive alternative.[104]

---

[100] 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 over the term should consult WIKIPEDIA, TALK: VIRAL LICENSE. http://en.wikipedia.org/wiki/Talk:Viral_license

[101] HERBERT HOVENKAMP 11 ANTITRUST LAW: AN ANALYSIS OF ANTITRUST PRINCIPLES AND THEIR APPLICATION (2d ed. 2005) ¶ 1906, p. 523.

[102] *Id.* at ¶ 1906, pp. 238-39; MARK A. LEMLEY & CHRISTOPHER R. LESLIE, CATEGORICAL ANALYSIS IN ANTITRUST JURISPRUDENCE, 93 *Iowa L. Rev.* 1207, 1221 (2008).

[103] Formally, 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 Prof. 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, 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” and instead argues in favor of upholding restraints that are “in the opinion of the joint venture partners, reasonably necessary for their success.” *Id.*

[104] HERBERT HOVENKAMP 11 ANTITRUST LAW: AN ANALYSIS OF ANTITRUST PRINCIPLES AND THEIR APPLICATION (2d ed. 2005) ¶ 1913; *Sullivan v. National Football League*, 34 F.3d 1091, 1112 (1st Cir.

We 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"[105] 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. Probably the simplest situation involves the vast majority of OS licenses that give named "stewards" discretion to issue amended terms.[106] Here, it should normally sufficient for the court to assert jurisdiction over the steward and order him or her to issue an appropriate less restrictive alternative. 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 overbroad license.[107]

Viral Economics. The Free Software Foundation ("FSF") invented GPL to encourage (less politely: compel) programmers to switch from CS to OS licenses. In its 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.[108] Programmers who wanted to use the modules would therefore have to adopt GPL themselves. The idea of unique

---

1994), *cert. Denied* 513 U.S. 1190 (1995); *Wilk v. American Medical Assn.*, 671 F.Supp. 1465 (ND Ill. 1987), aff'd, 895 F.2d 352 (7th Cir.), *cert. denied* 496 U.S. 927.

[105] HERBERT HOVENKAMP 2A ANTITRUST LAW: AN ANALYSIS OF ANTITRUST PRINCIPLES AND THEIR APPLICATION (2d ed. 2005) ¶ 325, pp. 5, 7.

[106] Practically all moderate and strong viral clauses include steward clauses. Examples include the Free Software Foundation's GPLv2 (*infra*, note 122) at ¶ 9, GPLv3 (*infra*, note 130) at ¶ 14, and LGPL (*infra*, note 116) at ¶ 6; Netscape's Mozilla License (*infra*, note 143) at ¶ 6; Apple Computer's Public Source License (*infra*, note 149) at ¶ 7; Sun's Sun Industry Standards License (*infra*, note 160) at ¶ 6; The IBM Public License (*infra*, note 152) at ¶ 7; the IBM Common Public License (*infra*, note 153) at ¶ 7; and the Eclipse Public License (*infra*, note 154) at ¶ 7. Among the licenses reviewed for this article, only the very minimal Apache License (*infra,* note 163) and licenses modeled on the Berkeley Software Distribution license (*infra,* note 166) fail to name a steward.

[107] ILAN CHARNELLE, JUSTIFICATION AND SCOPE OF THE COPYRIGHT MISUSE DOCTRINE AND ITS INDEPENDENCE OF THE ANTITRUST LAWS, 9 *UCLA Ent. L. Rev.* 167 (2002); BRETT FISCHMAN AND 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 (2002); DENNIS S. KARJALA, COPYRIGHT PROTECTION OF OPERATING SOFTWARE, COPYRIGHT MISUSE, AND ANTITRUST, 9 *Cornell J.L. & Pub. Pol'y* 161 (1991).

[108] 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." FREE SOFTWARE FOUNDATION, WHY YOU SHOULDN'T USE THE LIBRARY GPL FOR YOUR NEXT LIBRARY. http://www.gnu.org/software/chinese/www/why-not-lgpl.2002-09-26.html.

modules was probably reasonable in a world of individual programmers were usually incapable of duplicating large code bases on their own. However, the concept was always much shakier for commercial developers.[109] The reason, as FSF never tired of pointing out, was that "Proprietary software developers have the advantage of money."[110] For companies, the question was less whether a supposedly "unique" program could be duplicated – clearly it could[111] – but whether the profits from doing so were greater than those which 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 Section 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, non-viral 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-or-leave-it decisions for bundles containing multiple modules.[112] Here, shrewd license design can increase the number of

[109] FSF seems to have assumed that most commercial companies would continue to practice CS with or without GPL. 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 supplied).

[110] FREE SOFTWARE FOUNDATION, *supra* note 90.

[111] 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 headstart, *i.e.* that CS companies only learn about the OS module after it has been written. Today's companies monitor OS developments so closely that such surpises are rare.

[112] 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 (2004) In principle, therefore, 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 a significant antitrust problems given courts' admitted reluctance to second-guess software designers' architecture choices. *See, e.g., U.S.*

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.[113] 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. GPL solves this problem by using objective proxies based on the extent to which new modules copy or dynamically link to preexisting OS code.[114] 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 so-called Lesser General Public License ("LGPL")[115] – which permits dynamic linking and is therefore less infective – in cases where the normal GPL license would be ineffective or counterproductive.

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. This aggravates the already-large imbalance between OS and CS companies and reinforces the cartel effect. Second, software development is not an 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.[116]

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

---

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).

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

[114] 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 113.

[115] http://www.gnu.org/copyleft/lesser.html

[116] ALFONSO GAMBARDELLA & BRONWYN H. HALL, PROPRIETARY VERSUS PUBLIC DOMAIN LICENSING OF SOFTWARE AND RESEARCH PRODUCTS, 35 *Research Policy* 875 (2006).

whatever the market will bear. This, of course, is a formula for disaster: If every company does this OS will disappear entirely.[117]

Viral provisions block this outcome by forcing companies that use OS code to donate their improvements back to the community. The question is, how narrow can this restriction be and still ensure stability? 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 Prof. Vetter has noted, is that programmers have considerable technical freedom to move software from one module to another.[118] 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 the widely-used Mozilla license and its imitators. 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.[119]

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. 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 Prof. Hovenkamp has remarked:

> "[P]laintiffs cannot be permitted to offer less restrictive alternatives whose efficacy is mainly a matter of speculation…Proferred less restrictive alternatives should either be based on actual experience in analogous situations or else be fairly obvious. Tending to defeat such an offering would be the defendant's

[117] *Id.*; SUZANNE SCOTCHMER, OPENNESS, OPEN SOURCE, AND THE VEIL OF IGNORANCE, 100 *American Econ. Rev.* 165–71.. http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1524051; VON ENGELHARDT, *supra* note 41.

[118] VETTER, *supra* note 94.

[119] The license does not have to prescribe 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 "inventing around" costs provide the best measure of patent breadth. SCOTCHMER, *supra* note 53 at 105, 107-112.

> evidence that the proferred alternative has been tried but failed, that it is equally or more restrictive, or otherwise unlawful."[120]

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

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.[121] While the scope of this phrase has been much debated[122], it seems to include (a) files that include any verbatim portion of the preexisting code,[123] (b) entirely new modifications to that code,[124] (c) any interface files, compiler and installation scripts needed to run that code,[125] and (d) all "derivative work,"[126] a term that arguably incorporates the full reach of "derivative works" under US copyright law.[127] 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.[128] To the extent that viral terms apply, programmers must adopt GPLv2 for their own work and cannot charge royalties.[129]

[120] HOVENKAMP, *supra* note 102 at ¶ 1913b and pp. 343-44.

[121] GNU PUBLIC LICENSE VERSION 2 (hereinafter "GPLV2") at ¶ 0. http://www.gnu.org/licenses/old-licenses/gpl-2.0.html.

[122] *See, e.g.*, JASON B. WACHA, "TAKING THE CASE: IS THE GPL ENFORCEABLE?" 21 *Santa Clara Computer & High Tech. L.J.* 451, 488 (2005) (GPL's "based on" language presents "a thorny issue of interpretation).

[123] GPLV2, *supra* note 122 at ¶ 0.

[124] *Id.* at ¶ 0.

[125] *Id.* at ¶ 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.*

[126] *Id.* at ¶ 0.

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

[128] FSF has suggested that courts decide which links qualify "both on 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 GPL'S VIRAL EFFECT. 25 *J. Marshall J. Computer & Infor L.* 487, 493-4.

[129] GNU General Public License at ¶¶ 3, 6. http://www.gnu.org/licenses/gpl-3.0.html.

> *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.[130] Programmers who write such software must license it at zero royalties under GPLv3[131] and also supply enough "Corresponding Source" software for users to run the code.[132]
>
> *Oracle's Berkeley Database License.*[133] The scope of this license includes both preexisting OS code and "modifications."[134] Programmers whose work falls within this definition must make their source code available to users together with "any accompanying software that uses" it.[135]

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 mixed OS/CS markets could have "too much OS."[136] 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.[137] Finally, Linus Torvalds has repeatedly relaxed GPL so that programmers can write CS programs that make "normal system calls" on LINUX[138] and even insert CS "loadable kernels" into LINUX itself.[139] Despite this, the LINUX collaboration shows no signs of instability.

---

[130] *Id.* at ¶ 0.

[131] *Id.* at ¶ 2.

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

[133] http://www.oracle.com/technology/software/products/berkeley-db/htdocs/oslicense.html.

[134] *Id.*

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

[136] *See, e.g.,* FREE SOFTWARE FOUNDATION, *supra* note 97.

[137] 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? 21 *Berkeley Tech. L.J. 1421* (2006).

[138] 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, 311-312. 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 GNU GPL, 85 *B.U.L. Rev.* 1439, 1452. Conversely, Torvalds has argued that GPL does not reach preexisting software that has been modified to place calls on the LINUX kernel. GREG R.

Case 2: Weak Viral Terms. LINUX apart, "infectious terms have been GPL's least imitated feature."[140] 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 participate.[141] This may explain why commercial OS collaborations overwhelmingly choose so-called "weak copyleft licenses." Examples include:

> *Mozilla Licenses.* The "Mozilla Public License"[142] 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.[143] In practice, it is usually interpreted in ways that permit CS programs to include unlimited dynamic and static links to the underying software and also construct "larger" CS works that bundle OS and CS software together.[144] Covered software must, however, be made available to users as source code on a royalty-free basis.[145] Sun's "Common Development and Distribution License"[146] and "Sun Public License"[147] contain similar provisions.
>
> *Apple Public Source License.*[148] This license asserts Mozilla-type breadth but permits companies to sell CS modifications provided that the source code is supplied to users.[149] Here, the formal ability to license extensions is undercut by the requirement to distribute source code which makes it easier for consumers to

VETTER, 'INFECTIOUS ' OPEN SOURCE SOFTWARE: SPREADING INCENTIVES OR PROMOTING RESISTANCE? 36 *Rutgers L.J.* 53, 154-55 (2004).

[139] JOHN TSAI, NOTE: "FOR BETTER OR WORSE: INTRODUCING THE GNU GENERAL PUBLIC LICENSE VERSION 3," 23 *Berkeley Tech. L.J.* 547, 560 (2008).

[140] VETTER, *supra* note 120 at 152.

[141] For example, Sun decided not to use GPL because it wanted collaboration members to be able to create "larger works for commercial purposes." CDDL FAQS. http://www.openmediacommons.org/CDDL_FAQs.html.

[142] http://www.mozilla.org/MPL/MPL-1.1.txt.

[143] *Id.* at ¶ 1.9.

[144] *See, e.g.,* WIKIPEDIA. COPYLEFT. http://www.globalwarmingart.com/wiki/Wikipedia:Copyleft

[145] MOZILLA PUBLIC LICENSE, *supra* note 137 at ¶¶ 2.1 and 2.1.

[146] http://www.sun.com/cddl/cddl.html

[147] http://www.opensource.org/licenses/sunpublic.php

[148] http://www.opensource.apple.com/license/apsl/

[149] *Id.* at ¶ 2.2(c).

evade licensing restrictions on, for example, providing the software to unauthorized users. Similar requirements can also be found in Sybase's "Sybase Open Source License."[150]

IBM/Eclipse Licenses. The "IBM Public License"[151] "IBM Common Public License,[152] and Eclipse Foundation "Eclipse Public License"[153] all assert Mozilla-type breadth.[154] 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.[155] Furthermore, the Eclipse license does not extent to "separate modules" that do not qualify as "derivative works" under US copyright law.[156] While Eclipse concedes that the precise boundaries of this exception are unclear,[157] companies are clearly permitted to write CS licenses for "Eclipse plug-ins" that contain "100 %" new code and "implement functionality not currently in Eclipse."[158]

Sun Industry Standards Source License (SISSL).[159] This unusual license 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. On the other hand, companies whose products do not satisfy the Open Office standards must publish a list of all

---

[150] http://opensource.org./licenses/sybase.php at ¶ 2.2 (c).

[151] http://www.opensource.org/licenses/ibmpl.php

[152] http://www.ibm.com/developerworks/library/os-cpl.html. The license's viral terms are limited to "Contributions" to the program. They specifically do not include 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.

[153] http://www.eclipse.org/legal/epl-v10.html

[154] IBM PUBLIC LICENSE *supra* note 148 at ¶ 1; IBM COMMON PUBLIC LICENSE *supra* note 149 at ¶ 1; and ECLIPSE PUBLIC LICENSE, *supra* note 150 at ¶ 1.

[155] IBM PUBLIC LICENSE *supra* note 148 at ¶ 3(b)(iv); IBM COMMON PUBLIC LICENSE *supra* note 149 at ¶ 3(b)(iv); and ECLIPSE PUBLIC LICENSE, *supra* note 150 at ¶ 3(b)(iv).

[156] IBM PUBLIC LICENSE *supra* note 148 at ¶ 1(b); IBM COMMON PUBLIC LICENSE *supra* note 149 at ¶ 1(b)(ii);and ECLIPSE PUBLIC LICENSE, *supra* note 150 at ¶ 1(a).

[157] For example, the Foundation does not say how many links to, or copying of the underlying OS software triggers "derivative work" status. ECLIPSE FOUNDATION. ECLIPSE PUBLIC LICENSE (EPL) FREQUENTLY ASKED QUESTIONS. http://www.eclipse.org/legal/eplfaq.php ¶¶ 26.

[158] *Id.* at ¶ 27.

[159] http://www.openoffice.org/licenses/sissl_license.html.

> deviations and/or the modification's source code.[160] In either case, the obligation to disclose source code is much lower than for other Mozilla-type licenses.

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. Second, only the Mozilla license completely bars users from selling modified code under CS licenses. That said, all of the non-Mozilla licenses restrict this right, usually by requiring users to make the source code for their modifications available to users.[161] Because these terms make CS licensing more difficult, they are at least arguably related to stability.

Case 3: Minimally Viral Terms. Finally, many commercial projects contain only minimal viral terms. Examples include:

> *Apache License.*[162] This license formally reaches all software that is based on or derived from the underlying OS code.[163] 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.[164]

> *Berkeley Software Distribution (BSD).* BSD formally covers all "redistribution with or without modification."[165] However, licensees' obligations are limited to retaining certain disclaimers and notices.[166] This gives users an unrestricted right to license both original and modified versions of the underlying OS software commercially. Commercial versions of BSD include the "Cryptix General License"[167] and the "Intel Open Source License."[168]

[160] *Id.* at 3.1..

[161] Requiring users to disclose source code for their modifications almost certainly makes CS less attractive. Given a choice, almost all commercial CS companies withhold source code in order to discourage unauthorized use.

[162] http://www.apache.org/licenses/LICENSE-2.0 *see also* APACHE FOUNDATION, FREQUENTLY ASKED QUESTIONS. http://www.apache.org/foundation/licence-FAQ.html#My-License

[163] *Id.* at ¶ 1. The license gives no indication that it seeks to incorporate the meaning ascribed to "derivative works" under the US Copyright Act.

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

[165] http://www.opensource.org/licenses/bsd-license.php.

[166] *Id.*

[167] http://cryptix.org/LICENSE.TXT

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 serve individual clients. Here, CS business models are usually regardless of license terms. It would be more informative to find Apache or BSD-style licenses in markets where some CS transactions already occur. Apple's decision to make many of its software products available under Apache instead of the weakly viral Apple Public Source license[169] is a particularly intriguing example.

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.[170] 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 Services 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 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. At the same time, deliberate cartelization remains a plausible threat. This provides yet another reason for courts and policymakers to scrutinize GPL-style licenses.

[168] http://www.opensource.org/licenses/intel-open-source-license.php

[169] http://en.wikipedia.org/wiki/Apple_Public_Source_License

[170]*But see*, JOSH LERNER & JEAN TIROLE, THE SCOPE OF OPEN SOURCE LICENSING, 21 *Journal of Law, Economics and Organization* 20 (arguing that commercial firms may sometimes adopt GPL to attract volunteer programmers who would not otherwise participate).

VI. Government Innovation Policy: Taxation, Subsidies, Grants, and Procurement Policy

Section III has argued that the OS cartel effect suppresses quality competition and leads to a systematic imbalance between OS and CS companies. On the other hand, Section 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 quantity competition is likely to restore this balance. Relief, if it comes at all, requires government intervention.

The idea of government intervention has been widely debated for most of the past decade. At first, pro-OS scholars assumed a simple objective: If free software was good, then more would be better. This led to many different proposals for promoting OS through procurement preferences, taxation, subsidies, and other interventions.[171] Significantly, scholars who opposed these suggestions usually agreed that more OS was always desirable. Instead, they objected that OS volunteers were hardly ever motivated by money.[172] This implied that government initiatives based on traditional levers like taxation and/or government spending could do little to change behavior.[173] This academic discussion was followed by a similarly inconclusive debate in governments around the world.[174] While some promotion schemes were enacted, most were abandoned and new proposals dropped off rapidly after 2005. 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

---

[171] *See, e.g.,* LAWRENCE LESSIG, THE FUTURE OF IDEAS 247 (2001). 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.

[172] 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, 341, 344 (2003) (OS is "primarily developed by individuals who donate their time to work on projects that interest them" and "The 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 (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. Law & Tech.* 473, 481-83 (2003) (noting that most OS is motivated by non-monetary incentives and asserting that commercial OS investments "are likely to remain limited" because of free-riding).

[173] EVANS & REDDY, *supra*, note 133; p. 389; SCHMIDT AND SCHNITZER, *supra* note 145 at 498-99.

[174] Politicians 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 US multinationals (Costa Rica, Japan, Russia, Thailand), improving computer security (Costa Rica, Argentina, Iran) and reducing copyright piracy (Iran, Thailand). CENTER FOR STRATEGIC AND INTERNATIONAL STUDIES (CSIS), GOVERNMENT OPEN SOURCE POLICIES - 2008. http://csis.org/files/media/csis/pubs/0807218_government_opensource_policies.pdf. See *also*, JYH-AN LEE, *supra* note 145 at 101, 105.

proprietary processes and products."[175] 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. On the other, 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.

Purchasing Preferences. Given that governments already spend large amounts of money purchasing software, politicians often argue that redirecting these resources to promote OS is essentially costless. 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.[176] Furthermore, at least four countries have actually adopted such measures[177] and six more have adopted "preferences" for OS where its performance is comparable to CS alternatives.[178]

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, (b) prevent all-CS markets from evolving into mixed OS/CS states, and (c) systematically over-supply OS companies where mixed OS/CS states already exist. Traditional pro-OS purchasing preferences could encourage new OS entry. This would be irrelevant in case (a) but potentially helpful in case (b). On the other hand, we have already seen that OS companies in mixed markets are already oversupplied. This suggests that government procurement preferences for OS software would make problem (c) even worse than it is today.

[175] CSIS, *supra* note 147 at 1.

[176] 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 procument. *Id.*

[177] 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 requiremens or preferences in favor of OS. *Id.*

[178] Australia, Belgium, Brazil, China, Malaysia and Spain. *Id.* 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 ninety-six percent adoption rate suggests that something more systematic is at work. OPEN SOURCE OBSERVATORY AND REPOSITORY, FR: ALMOST THE ENTIRE PUBLIC SECTOR IS USING OPEN SOURCE (Sept. 30, 2009). http://www.osor.eu/news/fr-almost-entire-public-sector-is-using-open-source.

But if pro-OS preferences are a bad idea, why not adopt pro-CS preferences instead? True, the idea is politically unlikely. But this could change as the image of OS becomes steadily 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 therefore become more profitable. This would eventually lead to more CS entrants and dilute the cartel effect. However these gains would come at a price:

> *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 Fig. 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.[179] This could be a good thing if it means that there is no chance of overshooting the desired mix.
>
> *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 governemnt preferences will cause CS cmopanies to increase their investments, but not as fast as the value of the preference.
>
> *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.

Government spending is not, of course, limited to existing procurement.Government can also budget new expenditures to promote OS methods. Here, the most popular proposals have usually involved subsidies to OS firms, tax breaks, and direct purchases of new OS software.

Subsidies. Governments have launched a variety of initiatives to subsidize OS methods including grants and cash incentives to private sector OS adopters,[180] government-funded projects and alliances to promote OS methods,[181] government-funded training, support, and demonstration centers, [182] and government-funded OS workshops.[183] Since

[179] JYH-AN LEE, *supra* note 145. (estimating $17 billion government procurement market worldwide as of 2002)

[180] Singapore, Hong Kong, and Israel. Israel also offers funds for OS start-up companies. CSIS, supra, note 147.

[181] Cambodia, China, Czechoslovakia, Israel, South Africa, South Korea, Japan, Netherlands, Philippines, Thailand, and Vietnam. *Id.*

[182] Brazil, Malaysia, Pakistan, Western Australia, and Spain. *Id.*

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. 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 bound to crowd out some private investment.

Once again, the deepest problem relates to fine-turning, *i.e.* knowing when to stop the subsidies once a mixed OS/CS market exists. Section III has argued that OS collaborations should follow something like the Five Effort rule – *i.e.* that the number of OS companies should be comparatively small. 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.

Tax Policy. Economics textbooks routinely praise the non-distortionary benefits of tax policy, *i.e.* using 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 (CS) companies less (more) profitable and shifting the OS/CS ratio toward the fifteen to twenty percent target suggested by the Five Effort Rule.

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

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 develop more OS code, why not purchase it directly? To some extent, government grants do this already. This usually happens informally when grant agencies let individual faculty – most of whom are ideological predisposed to OS – decide how to license their government-funded software. Furthermore, many US grant agencies – notably NIH – now require grant applicants to provide "dissemination strategies" for sharing their data and discoveries with the wider world.[184] OS licenses provide a particularly simple way to meet these requirements.

[183] Brazil. *Id.*

[184] *See, e.g.*, NATIONAL INSTITUTES OF HEALTH, NIH DATA SHARING POLICY AND IMPLEMENTATION GUIDANCE. http://grants.nih.gov/grants/policy/data_sharing/data_sharing_guidance.htm.

Indeed, the UK has taken this logic one step further by adopting interim regulations that make OS licenses the “default position” for government-funded software.[185] Finally, many governments fund projects, institutes, alliances, and consortia whose missions include writing OS software.[186]

Not surprisingly, purchasing still more OS from the private sector makes the usual imbalance in mixed OS/CS industries even worse. On the one hand, government contracts make OS companies even more profitable. This increases their numbers and makes the OS-to-CS firms even less favorable than it was before. 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 Fig. 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.[187] 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 which 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, therefore, companies would have to propose software that consumers actually wanted.[188]

---

[185] The regulations provide: “[I]if 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 UK-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.’” CSIS, supra note 147.

[186] China, Finland, Japan, South Korea, France, India, Slovakia, Spain, Thailand, Venezuela, and Vietnam. *Id.*

[187] SCHMIDT & SCHNITZER, *supra* note 145 at 495 (“…a government-sponsored research lab does not face the constraints of the market and has much less incentive to focus on consumer needs and cost-efficiency.”)

[188] STEPHEN M. MAURER & SUZANNE SCOTCHMER, *Procuring Knowledge in* 15 ADVANCES IN THE STUDY OF ENTREPRENEURSHIP, INNOVATION AND GROWTH1 (Gary Libecap ed., 2004).

VIII. 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. We have argued that this effect usually stops OS output far short of its theoretical potential.

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

Table 1: Top 15 Eclipse Contributors[189]

Company (Membership Status)	**Number of Commits since 2001**	**Number of employee programmers involved since 2001**	**Number of Modules supported since 2001 (Leading Contributor)**	**Company Profile**
1. IBM (Strategic Member)	3,521,216 (39%)	318 (26%)	80 (44)	Hardware, software, and consulting services.
2. Italio Inc. (Supplier Member)	375259 4%	10 1%	15 (7)	Business management software and services including Intalio|Designer, an eclipse-based integrated development[190]
3. Oracle (Strategic Member)	339513 (4%)	41 (4%)	18 (5)	Business management software specializing in information management.
4. Actuate Corp. (Strategic Member)	188890 (2%)	53 (4%)	7 (3)	Business management and reporting software
5. Itemis AG (Strategic Member)	139506 2%	13 1%	11 (6)	Model driven software development (MDSD) and Eclipse based tool chains.
6. OBEO (Strategic Member)	125852 1%	14 2%	20 (6)	Software tool vendor and a consultant
7. Red Hat, Inc. (Solutions Member)	124483 (1%)	12 (1%)	40 (5)	Consultant and seller of pre-packaged "distributions" assembled from LINUX modules.
8. Sonatype (Strategic Member)	102650 (1%)	1 (0%)	8 (0)	Sonatype provides a development infrastructure platform standard development infrastructure.[191]

[189] Statistical data for total commits and programmers obtained from Eclipse Foundation Dashboard at http://dash.eclipse.org/dash/commits/web-app/summary.cgi. Company profiles are drawn from the Eclipse member pages at http://www.eclipse.org/membership/. Additional company information for IBM (http://en.wikipedia.org/wiki/IBM), Oracle (http://en.wikipedia.org/wiki/Oracle_Corporation), Actuate (http://en.wikipedia.org/wiki/Actuate) is drawn from Wikipedia.

[190] Blooomberg Business Week:"Intalio Inc.", http://investing.businessweek.com/research/stocks/private/snapshot.asp?privcapId=141317

[191] Web Page: "About Sonatype" http://www.sonatype.com/about/index.html

9. Tasktop (Solutions Member)	93410 (4%)	4 (1%)	10 (1)	Task management solutions for Eclipse.
10. Soyatec (Solutions Member)	93340 (1%	4 (1%)	7 (3)	Software company and a major Eclipse solution provider.
11. Borland Software (Former Board Member)	89549	6 ()	5 (2)	Software vendor
12. Intel Corp. (Solutions Member)	83924	22 ()	14 (3)	Solutions Microchip fabrication, software products, professional services.
13. Innoopract (Strategic Member)	75411	11	11 (4)	Strategic Member Support services for companies developing applications using Eclipse
14. Thales (Associate Member)	68522	2	9 (3)	Associate Member N/A
15. SOPERA GmbH (Strategic Member)	54608	10	5 (1)	Service provider dedicated Web solutions using Serice Oriented Architectures.