The Efficiency of Software Development Models: Proprietary vs. Free/Open Source

Prepared for
Professor David R. Kamerschen
Economics 5900
Spring Semester, 2009

Prepared By
Matthew J. Slotten
157 Woodrow Street

Athens, GA 30605

770-262-7538

mattslotten@gmail.com

April 14th, 2009

Abstract

The market for software is in many ways distinct from that of any other good due to software's strong network effects, imperfect public-good nature, and complexity. As a result, the free market has two primary means of delivering software to consumers: the proprietary method, and the free/open source method. These two contrasting methods of software production may appear to be mutually exclusive, and many believe their coexistence to be unsustainable. This study investigates the theoretical and empirical literature regarding the rationale and motivation for, and welfare impacts and overall efficiency of, each software development model and how they interrelate. We can conclude from this study that these two models are in fact not mutually exclusive and in some cases complementary, provided certain market conditions are met. We can also conclude that the open source software development model is in fact consistent with economic theory and is thus a sustainable method of software production.

The Efficiency of Software Development Models: Proprietary vs. Free/Open Source

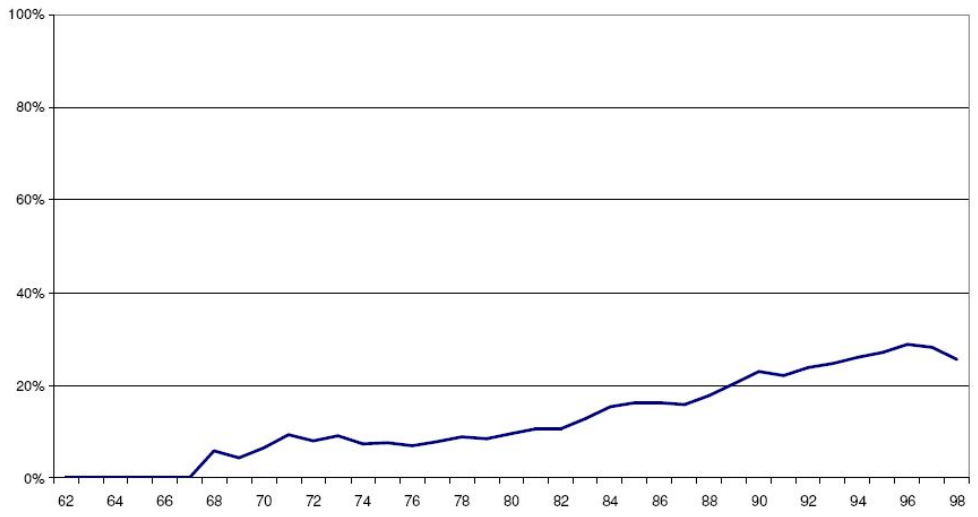
The advent of modern computer technology in the mid-1960s and its continued high growth has generated a substantial demand for software to run on these systems. In an effort to meet this demand, two models of development for software production have arisen. The first is the proprietary model, in which a rent-seeking private firm employs labor and capital to produce a software product in which the source code is not revealed so as to claim the software as intellectual property. When most people think of software production, they are likely thinking of this method, with Microsoft being a key example. In the case of proprietary software, the firm producing the software must implement means to counter software's public-good nature. Practices such as software licenses, software as a service, and software patents are used to transform software into an excludable and rival good. Pre-packaged software is the most common type of proprietary software, where the software is available in a packaged form, not tailored or custom written for one particular customer's needs. The second model is known as the free/open source model, in which individuals, and in some cases firms, collaborate to create software, primarily distinguished from the proprietary model by having source code freely available to the public under certain conditions. Open source software relies entirely on volunteer contributions, as property rights are given to the public in general. Notable open source projects include the Linux operating system, the Apache web server, and the Mozilla Firefox browser. The open source model produces software as a pure public good, leading many to question the sustainability and incentives that allow this model to efficiently produce software. Given the stark differences of these two models, many question how these models can coexist and argue that in time, one will extinguish the other through Schumpeterian creative destruction. After reviewing the literature and applying basic economic principles, I reach a very different conclusion. Competition between open source software and proprietary software is sustainable in the long run, and their coexistence is welfare improving.

Software: A Brief History and Overview

Software development has a tradition of cooperative development, allowing projects to benefit from different contributors' comparative advantages. Larger software projects often require multiple developers, as different aspects of software projects, such as user interfaces, documentation, and different functionalities, require different skill sets. Software can be packaged as either "open source" or as "pre-packaged binaries." In the case of open source software, the code that allows the software to function is freely distributed and readable by the user. In the case of pre-packaged software, the code is converted into machine-readable code, not easily interpreted or modified by users. Most commercial software vendors today release software in the pre-packaged binary format, so as to preserve intellectual property rights on the software they produce. Josh Lerner and Jean Tirole divide software development into three distinct eras (2006, p. 4).

The first era of the 1950s through 1970s is characterized primarily by development in academia and central research facilities. As James Bessen notes, owning a computer prior to the 1970s typically required most users to either self-develop software or contract custom software development (2005, p. 8). In the context of computing history, this makes sense, as single-person computer systems were available to people only in academic or research institutions, as these systems were still too expensive to be owned by private individuals. Lerner and Tirole note that in this first era, most software development had a loosely structured open source nature, without clearly defined property rights where individuals would freely swap code and standards (2006, p. 4). These poorly defined property rights soon resulted in legal implications for both IBM and AT&T in the late 1970s and early 1980s (respectively) when they began claiming property rights on computer operating systems. Not until the 1970s did a market for pre-packaged software emerge as a result of the expansion in the low-cost personal computer market, as illustrated in Figure 1.

Figure 1: Packaged Software Share of All Software Investment



Source: Grimm and Parker (2000) p. 38

The second era of software development spanned the 1980s into the early 1990s. This era of software development is characterized by high growth for commercial software development firms, giving rise to now software giants such as Microsoft, Oracle, and SAP. As Bessen notes, competing software firms began participating in "feature wars," in hopes of gaining market share and a dominant position in the market (2005, p. 8). Software became increasingly complex as innovations in microprocessor technology permitted larger and more robust applications. Despite the rapid increase in available feature-rich pre-packaged proprietary software, it did not crowd out self-development solutions. As illustrated in Figure 1, roughly 30% of total software investment is devoted to pre-packaged software. This figure indicates that pre-packaged software has failed to meet the needs of a significant portion of the market. While this era saw the rise of commercial software vendors vying for market share, it also gave rise to The Free Software Foundation, which introduced a general public license, or GNU, a cornerstone of open source software development today. Although the formal definition of the GNU provided a more structured approach to open source software projects, developers still lacked a low-cost means of collaboration until the Internet became widely available in the early 1990s.

The third and current era of software development has seen a strong acceleration of open source software development as a result of inexpensive and widespread availability of internet access. As Alfonso Gambardella and Bronwyn Hall point out, the increase in computer networking and the rise of the internet essentially lowered the marginal cost of communicating codified knowledge to zero, better facilitating the open source model of software development (2005, p. 4). Well-known open source projects such as the Linux operating system and the

Apache web server were both developed in this era. Online communities supporting open source development have proliferated, such as SourceForge.net and Tigiris.org. The proliferation of the internet has also given rise to increased software piracy, allowing individuals to illegally download pre-packaged software without paying for it, leading commercial software firms to implement anti-piracy measures such as requiring unique license keys and product activation before installing or using the software, and increased consumer education, all in attempts to defeat the public-good nature of software.

Software, in its basic form, exhibits qualities of a traditional public good. A public good is a good whose consumption is both non-rival and non-excludable. As one user's use of a software application on one system does not inhibit another user's use on another, the consumption of software can be considered non-rival. Furthermore, a software application can be easily copied or downloaded from one computer to another at no cost to the user, making the consumption of the application non-excludable. As discussed later, commercial firms implement measures into their pre-packaged applications so as to defeat their public-good nature and allow them to earn economic profits from their software.

Another key characteristic of software as an economic good is that its consumption carries with it significant network externalities, which occur when the value of a good to a potential consumer is determined in part by the number of consumers already owning or using that good. Many software applications carry with them certain protocols, which they use to communicate with other systems and applications. As more users adopt a particular protocol, it becomes the de facto standard, forcing all future applications to use the established standards to maintain compatibility, thereby increasing market size. Bruce Endries's article, "Tech G.P.: All

Web Browsers Not Created Equal," suggests one such example in the case of internet browsers. Although Microsoft's Internet Explorer dominates the market at around 67 percent today, it does not adhere to industry standards, set in large part by Netscape Navigator, the first widely-available browsers eventually driven out of the market by Microsoft. These standards are maintained by several organizations, including the World Wide Web Consortium (W3C), which outline how browsers should interpret certain code (2009). Because Microsoft's browser is not entirely standards-compliant, many web page developers either have to accommodate Internet Explorer's nuances, or the full capability of their website is not available to Internet Explorer's users. Bessen suggests that in order to overcome compatibility issues, many proprietary software firms release Application Programming Interfaces (APIs) to allow firms to customize how their current systems interact with the new pre-packaged product (2005, p. 4).

The Nature of Proprietary Software

The market structure for proprietary software can be generally characterized as monopolistic or oligopolistic with a small number of competitors. This market structure can be attributed to a number of factors. The first, as Bessen suggests, is that due to the complex-good nature of software in terms of features and compatibility, there exists no perfect substitute, and few if any imperfect substitutes, for software products (2005, p. 3). While different software products may perform similar tasks, such as word processing, they may do so in different ways, using different interfaces, and in different file formats. The second factor can be summarized in Bitzer's point that software production is dominated almost entirely by fixed costs, with little or no variable costs, and market segments for complex software products tend to produce a natural monopoly where economies of scale result in a barrier to entry for other firms (2004, p. 370). A good example of this is Microsoft's Windows operating system in the PC x86 architecture market; no other firms have successfully produced and marketed a pre-packaged operating system to compete with Windows. Lastly, proprietary software firms rely on patents to protect their software as intellectual property and maintain monopoly power, preventing potential market entrants from producing competitive products.

Software patents hold an important place in proprietary software production. Like any other patent dependent industry, proprietary software producers rely on the ability to patent their software to set monopoly prices. Lerner and Tirole have observed cases where individuals and companies do not produce software themselves, but instead simply hold software patents to collect economic rents from major software vendors (2004 p. 25). Copyrights and patents for software can be registered without revealing the source code behind the patents, allowing firms to maintain intellectual property rights. Software development as a whole may suffer from the "anti-commons" problem, where the patenting of software may lead to lower research productivity and cripple inventive activity, eventually leading to slower economic growth. While patents allow proprietary software developers to maintain monopoly power in the production of the software, developers must also attempt to overcome the free-rider problem in consumption of their software, something not necessarily accomplished inexpensively.

Since software in itself exhibits public-good characteristics, proprietary software developers have implemented a multitude of measures to overcome its non-rival and non-excludable nature. One such measure to make the pre-packaged software excludable is to simply not release its source code and only sell the binary executables of the software. Since users cannot read the source code, they cannot easily redistribute the software in different markets, otherwise known as arbitrage. Another measure implemented by proprietary firms which made the consumption of the software rival was to require users to enter a unique license key before installing the software. Prior to the widespread use of the internet, product keys were shipped in the packaging, so users were required to have the original packaging to install the software. However, this method proved only partially effective, as there was no way to verify that each license key entered was in fact unique. With the proliferation of the internet came another dimension of complexity for issuing license keys. Users could simply upload the software to other users along with the license key, allowing other users to pirate the software without having to pay for it. To counter the problem of piracy, proprietary software vendors implemented in their software the ability for the software to communicate with a central server, verifying that the entered license key was indeed unique. Some proprietary firms issue a hardware USB "key" or dongle that must be inserted into the computer at the time of installation. Michael Stolpe asserts that the hardware key method is in fact the most secure technical measure in software intellectual property protection. Stolpe has also found instances where software distributors lower the costs of obtaining license keys, so as to reduce the monetary incentive to pirate the software (2000, p. 28).

The Nature of Free and Open Source Software

At first glance, the open source software development model may seem paradoxical. Open source software is a public good, supported entirely by volunteer contributions. This suggests that open source software may be under-provisioned due to the free-rider problem.

Volunteer contributions seem to counter economic intuition that private agents would put forth sufficient effort to develop a public good without clearly defined property rights. Despite this, there still exist a multitude of successful OS projects, some are arguably superior to pre-packaged software.

Maurer and Scotchmer suggest that there must be a number of intrinsic and extrinsic incentives for individual volunteers to contribute. Extrinsic motivations require an "audience," where programmers aspire to better their reputation among the open source community or build their egos. Intrinsic motivations, in contrast, do not require an audience, where programmers contribute in an effort to express creativity, be part of a group or team, pursue educational purposes and experience accomplishment (2006, p. 12). Hakim Wafa Orman suggests in her article, "Giving it Away for Free? The Nature of Job-Market Signaling by Open Source Software Developers," that many developers may choose to release their code as open source instead of trying to bring it to market simply because the expected returns would not cover the cost of obtaining patents, marketing the software, and implementing anti-piracy measures (2008, p. 2). Lerner and Tirole emphasize the role of job-market signaling of OS contribution, as code is easily attributed to the individual who contributed it (2004, p. 16). Further, many individuals contribute to open source projects as a result of the ideology behind open source and in opposition to proprietary software. Maurer and Scotchmer found that open source contributors are overwhelmingly young male and single (Ghosh et al., 2002, Lakhani and Wolf 2005), with males comprising roughly 98% of all open source contributors (Henkel and Tins, 2004) (2006, p. 13).

Not all contributions are from individuals in their free time. Bessen notes that about half of the entire development effort of open source software is conducted by programmers at work with the knowledge of their supervisors (2005, p. 1). Private firms play an important role in open source development, as is the case with HP, Novell, and IBM. These commercial enterprises previously produced a UNIX variant operating system, but have since switched to support Linux, an open source UNIX variant. Bitzer suggests that this switch can be attributed to two reasons. The first is that the further development of Linux is cheaper than the incumbent's UNIX variant because the incumbent does not bear any of the development costs, as they are borne by volunteer contributors. The second is that since Linux is freely available and can be freely adapted to one's needs, using it does not force the incumbent firm to depend on another enterprise to further develop the operating system (2004, p. 370). These firms instead earn profit by supplying ancillary services, such as training, consulting, and support, or by selling complementary hardware. Many private firms may choose to use and contribute to open source software because it is adaptable to their own needs, as opposed to pre-packaged software, which is welfare-improving.

Open source licensing plays an important role in facilitating contribution and preserving the ideology of OS, maintaining that open source software should be free to use, free to modify, and free to redistribute. Maurer and Scotchmer highlight the social psychology perspective that volunteer incentives to contribute will be reduced if there is a widespread perception that third parties are profiting from the community's efforts (2006, p. 14). While there are sundry different open source licenses in use today, the two most pervasive ones are the Berkley Software Distribution (BSD) and the GNU General Public License (GPL). Lerner and Tirole outline the nature of each license. The BSD license is the most permissive license, permitting anyone to use the source code and redistribute it for a fee without making the source code publicly available. This places BSD licensed code closer to true public domain. The GPL license is more restrictive, requiring that all modifications be made freely available and insisting that others who use the redistributed code do so as well. The GPL license also requires that all contributed code must be licensed on the same terms (namely, GPL). Licenses such as the GPL are known as copyleft licenses, where they seek to keep intellectual property free and accessible, in contrast to copyright licenses (2004, p. 6).

While open source licensing aims to negate agents' ability to free ride on others' contributions in attempts to earn economic profits, no formal process inhibits free riding in terms of production of open source code. Maurer and Scotchmer illustrate this point well in their statement, "If ideas are not scarce – that is, if any good idea is likely to be had and implemented by someone else – it is tempting to let someone else bear the development cost" (2006, p. 23). In other words, open source development may be under provisioned because contributors who are otherwise willing to write code may wait in hopes that someone else will write it first. This phenomenon is essentially the opposite of patent incentives, where agents have an incentive to develop as quickly as possible, so as to win intellectual property rights. Game theorists find equilibria with both a pure strategy, where some users always contribute and some users always free ride, and with a mixed strategy, where each user contributes with some probability and development sometimes fails. Nonetheless, the number of open source projects is large and still growing, offering evidence that open source software development is sustainable in the long run.

Competition between Proprietary and Open Source Software

Since software is subject to substantial network effects and de facto standards, its utility varies with adoption and compatibility between platforms. As a result, proprietary firms experience little incentive to make their software products compatible with open source software in attempts to lock in their customers to using complementary revenue-generating products. In contrast, open source developers have a strong incentive to make their software compatible with existing standards, permitting the open source software to work in a network comprised of proprietary technologies, thereby increasing its adoption rate. An example of this is Samba, an open source software application that permits Unix-like systems to share files and print services with proprietary operating systems such as Microsoft Windows and Apple OSX. There exists no similar application developed to run on either Windows or OSX. Similarly, as Dalle and Jullien point out, it is now possible to emulate Windows on Linux machines, allowing Linux users to install Windows software on their systems, but no application exists to do the same on Windows machines (mimeo, p. 6). In her paper, "The Effects of Compatibility on Competition between Proprietary and Open Two-Sided Platforms," Wafa Hakim Orman uses a two-sided platform to provide a framework for studying operating systems. Orman finds that in some cases, proprietary firms may find it profitable to increase the compatibility of their software with open source software, though in most cases these firms lack the necessary incentives to do so (2008, p. 2). Increased compatibility between proprietary software is welfare improving, particularly for consumers, as it increases consumer choice and improves competition among software producers. Assuming open source software can continue to incorporate established proprietary standards, network effects alone may not be enough to extinguish competition.

Product heterogeneity plays an important role in the market for software. Software's complex nature permits producers to differentiate their product easily, including interface design, features, and standards. The simple fact that proprietary firms can sell their software at higher prices than open source alternatives indicates that product heterogeneity exists, at least to some extent. As a result, the market structure for many proprietary firms can be characterized as monopoly markets; however, the emergence of open source operating system has transformed many of these non-contestable markets into oligopoly markets. Since open source software is developed entirely by volunteer efforts, and the costs of development are carried by them, open source projects do not face the barrier to entry of decreasing average costs that proprietary firms do. In this respect, the open source mode of development may more closely reflect the competitive result: since the marginal cost of producing software is zero, its price should be zero. Open source software can therefore enter markets where other proprietary firms cannot and exert downward pressure on prices by offering consumers alternatives. Bitzer proposes using a Launhardt-Hotelling model to illustrate this effect, where the open source firm does not bear any R&D costs, while the proprietary firm bears substantial R&D costs. He postulates that price pressure on the incumbent firm is dependent on the perceived heterogeneity of its product (2004, p. 371). The more consumers see the open source alternative as a substitute, the further the incumbent firm will have to depress price, or further differentiate its software from the open source software. In this respect, open source software helps to restore competition to traditionally uncompetitive software markets.

While open source software does not present capital costs to consumers, it is unlikely that it will drive proprietary software vendors out of the market simply due to cost benefits. In many cases, open source software may in fact be costlier to consumers than equivalent prepackaged software due to non-capital costs. These costs may consist of transaction costs imposed as a result of network effects, difficulty of use and installation, and the unavailability of compatible programs. Because open source software is provisioned through volunteer development, contributors often choose to invest their efforts in projects that showcase their abilities rather than maximize consumer welfare. As a result, some aspects such as attractive user interfaces, product documentation, and technical support are under-provisioned in open source software, which in turn raise consumers' costs. Proprietary software-producing firms, on the other hand, have a strong incentive to appeal to consumer needs and therefore will often provide substantial technical support and product documentation along with their product, or as a two-part tariff. Dalle and Jullien suggest that because of Linux's lack of user-friendly interface and technical support, it failed to penetrate the consumer and PC market, but succeeded in the server market, where graphical user interfaces (GUIs) are not as highly valued (mimeo, p. 9). Proprietary firms may therefore be able to charge a price above marginal cost, which should be close to zero, and below the customer's perceived cost of implementing an open source solution, thereby earning a positive economic profit and maintaining market share.

The sustained competition between open source software and proprietary prepackaged software presents significant benefits to consumers. One obvious benefit is an increase in consumer choice. Through this increase in choice, consumers can maximize utility by selecting the software application that best fits their needs. Proprietary software typically offers easier to use interfaces, more features, and more thorough technical support than does open source, but at the cost of the customization. Open source software, on the other hand, offers easier customization, higher compatibility, and lower prices. Another benefit to consumers achieved through competition is a reduction in deadweight loss and an improvement in consumer surplus attributed to the fact that open source software has driven proprietary software prices down, closer to the competitive level. Lastly, competition between open source and proprietary software has fostered faster innovation within the industry, as open source software is better equipped to build on the ideas of the many, and proprietary firms are forced to keep up, thereby benefiting consumers.

Conclusion

Software's complexity has shaped private mechanisms to counter this obstacle, namely the proprietary and open source software development models of production. Although each model of software development differs fundamentally, in terms of incentive, market structure, and property rights, their coexistence is both sustainable and welfare improving. Open source software on its surface may seem perplexing to economists, after they evaluate software as an economic good and the incentives behind open source production; it is obvious that open source software does adhere to traditional economic principles. Open source software development has transformed previously monopolized markets into more competitive ones, consumers now have more choice, and network effects are overcome. We can expect to see more high growth in the software industry, driven in large part by this competition. Proprietary firms will likely see some of their market share further slip away to open source projects, but will likely maintain a dominant position in certain software markets. In the near future, we may observe a hybridization of these two models, which may be a subject of future research.

References

Bessen, James, “Open Source Software: Free Provision of Complex Public Goods,” July 2005, http://www.researchoninnovation.org/opensrc.pdf

Bitzer, Jurgen, "Commercial versus Open Source Software: The Role of Product Heterogeneity in Competition," Economic Systems, Elsevier, December 2004, 28(4), 369-381.

Dalle, Jean-Michel, and Jullien, Nicolas, “Open-Source vs. Proprietary Software,” mimeo, 16p., http://opensource.mit.edu/papers/dalle2.pdf

Endries, Bruce, “Tech G.P.: All Web Browsers Not Created Equal,” The Daily Star, 6 March 2009.

http://www.thedailystar.com/lifestyles/local_story_066043012.html

Gambardella, Alfonso, and Hall, Bronwyn H., “Proprietary vs. Public Domain Licensing of Software and Research Products,” National Bureau of Economic Research Working Paper w11120, February 2005,

http://www.nber.org.proxy-remote.galib.uga.edu/papers/w11120.pdf

Grimm, Bruce and Parker, Robert, “Recognition of Business and Government Expenditures for Software as Investment: Methodology and Quantitative Impacts, 1959-98”, Bureau of Economic Analysis, mimeo, 2000.

http://www.bea.gov/papers/pdf/software.pdf

Hakim Orman, Wafa, “Giving it Away for Free? The Nature of Job-Market Signaling by Open Source Software Developers,” B.E. Journal of Economic Analysis and Policy: Advances in Economic Analysis and Policy 8, January 2008, no. 1.

____________, “The Effects of Compatibility on Competition between Proprietary and Open Two-Sided Platforms,” March 2008, http://ssrn.com/abstract=1129275

Lerner, Josh, and Tirole, Jean, "The Economics of Technology Sharing: Open Source and Beyond," National Bureau of Economic Research Working Paper 10956, December 2004,

http://www.nber.org/papers/w10956

Maurer, Stephen M., and Scotchmer, Suzanne, "Open Source Software: The New Intellectual Property Paradigm," National Bureau of Economic Research Working Paper 12148, March 2006,

http://www.nber.org/papers/w12148

Stolpe, Michael, "Protection against Software Piracy: A Study of Technology Adoption for the Enforcement of Intellectual Property Rights," Economics of Innovation and New Technology, 2000, 9, no. 1, 25-52.