Competition between open-source and proprietary software: the (LA)TEX case study.

Alexandre Gaudeul *
Universities of Toulouse and of Southampton

September 17, 2004

Abstract

The paper examines competition between two development models, proprietary and open-source ("OS"). It first defines and compares those two models and then analyzes the influence the development of one type of software has on the development of the other. The paper is based on the (LA)TEX case study (Gaudeul (2003a)).[1] In that case study, the features, users, and patterns in the development of the (LA)TEX software were compared to its proprietary equivalents. The models that are presented in this paper describe some aspects of the strategic interactions between proprietary and open-source software. The paper shows that they cannot be analyzed independently; the decisions of one class of agents (OSS developers) are affected by those of the other class of agents (private entrepreneurs).

1 Introduction

Software and related services account for about half of the information technology expenditures of firms. Sales of packaged software generated $108 billion in revenues in 1997 in OECD countries, those revenues growing at 11% per year since 1990. (OECD Information Technology Outlook 2000) Software markets have attracted interest from a public policy point of view (see Katz and Shapiro (1999)).

A trend in recent year has been for companies to use open-source software ("OSS") to reduce their dependence on software and hardware vendors and their bundled service offerings. There is little economic analysis of the way that trend will affect the software industry. Existing work analyzes traditional and open-source ("OS") development practice separately, while the present paper shows that both cannot be analyzed independently; the decisions of one class of agents (OSS developers) are affected by those of the other class of agents (private entrepreneurs) as they compete for users but for different motives and following different rules.

*I would like to acknowledge the support of the CNRS, Jacques Crémer at the University of Toulouse, Hal Varian at the SIMS school in Berkeley and Robin Mason at the University of Southampton. I would like to thank all the participants in the TEX project that I interviewed. This paper was directly inspired by their reflections on the TEX community's own culture, history, future and organization.

It is important to understand how OSS affects welfare, as the governments, which are both important and leading investors in information and communication technology, must choose where to source their software needs and what type of protection software innovation should be allowed. Gaudeul (2004) deals with the question of what software license terms are best for welfare, while Crémer and Gaudeul (2004) or Lerner and Tirole (2004) explain further the interrelation between public policy and innovation in the software industry. Furthermore, it is also important for professionals and proprietary developers to understand how to position their product in the face of competition from OSS.

The main insights in the model are based on the study of the (LA)TEX open-source software (Gaudeul 2003a). Donald E. Knuth developed that typesetting software in the late 70s, released it with an open-source license and it has become a reference in scientific publishing. TEX and LATEX, a set of macro based on TEX, are now used to typeset and publish much of the world's scientific literature in physics and mathematics. Both software will be collectively referred as (LA)TEX. Appendix *E* presents the history of the development of TEX.

In this paper, I look at competition between open-source and proprietary software. The model builds upon some major differences between open-source and proprietary development:

- It is more difficult to achieve coordination and agreeing to goals in OSS development compared to proprietary development. This is illustrated by the (LA)TEX case study as well as by previous observations from the software engineering literature. This translates in the model as some probability that the development of some features will be duplicated when it is done OS because there is some chance that developers with the same needs do not meet and end-up developing the same features independently.

- Developers do not derive the same economic surplus from software development than the average user. This translates into different utility functions, the first, that of developers, not valuing the development of an user interface for the software, the second, that of the users, valuing the user interface as a way to ease installation and learning of the software. That difference can also be seen as a difference between experienced users of the software, who will therefore not care about an interface even though it could have helped them in the past to get to know and use the product, and novice users who have to decide whether to use the software or not.

- Proprietary software firms and individual open-source developers have different objectives. OS developers work under a license that prevents them in practice to make profit based on the selling of code they developed for an open-source project, so that they will develop features that have value to them, not necessarily to the market. This also means that the OSS price will be zero. Entrepreneurs on the other hand will try to maximize the market value of the software and sell it at a positive price. This paper also introduces another cause for the difference in objectives between OSS developers and entrepreneurs: the bodies ruling over

the direction of OSS development are dominated by the original set of users of the software, which means new or potentially new users find it difficult to influence the dynamics of the software's development. Entrepreneur on the other hand may be faster in spotting new market opportunities.

The rest of the introduction motivates the model further and presents some observations about the dynamics of OS vs. proprietary software competition.

The first part of the introduction exposes the way OSS organization achieve (or not) coordination between OSS developers, and how they choose their development objectives. In the case of TEX, this leads to the delineation of different stages in the OSS organization's life.

The second part of the introduction shows how difficult it is for an OSS organization to develop an interface, and shows how in the case of TEX private efforts by proprietary companies provided the first easy to use versions of the software.

The third part of the introduction shows how users and developers' needs are aligned in the choice of the OSS development objectives, not, as is stated too often, because users are developers, but because developers do not differ too much from the standard user, and because they understand the need to develop popular software to benefit from network effects.

The fourth part analyzes the patterns of competition between OS and proprietary software. They serve different users over time as each type of software develops in reaction to competition from the other side. Proprietary versions of the software serve to popularize the OSS on which they are based, or serve as inducers into the use of OS versions.

The paper is then divided in three parts, each one using the same framework:

In the first part, the lower quality of the user interface in open-source development translates into OSS serving either low or high value software users (depending on whether the OSS came to develop less or more features, respectively, than the proprietary software firms). There are also cases where the proprietary software retains a monopoly position on the market. When the proprietary software offers more feature than the OSS one, it will set a high price and users who cannot afford it will use the OSS even though it is more costly to learn. When both have the same features, then the proprietary firm will set its price at the value of its interface and gain the whole market. When the OSS has more feature, then some users who value the software's functionalities the most will forfeit the proprietary firm's interface and use the OSS instead.

In the second part, network effects are introduced. Network effects play a role when choosing software: users want to use software that others use. This is because they are concerned with adopting a widely used standard (as in the case of TEX and any document processing software, where the ability to exchange documents is crucial), or because they believe this will ensure continued development and maintenance of the software (that argument is particularly valid for OSS). Network effects are shown to lead more often to the development of a full-featured OSS as developers take into account the increase in market share from developing more features. This has an ambiguous

effect on the entrepreneur's decision, as she will more often be discouraged from developing a proprietary product, but when she does, she will make more profit and have a larger market share than when network effects were not present.

In the third part, users are divided into two groups, those whose utility functions for the software's features coincide with that of OS developers, and those who value its features differently. This leads to a differentiation between the features of proprietary and open-source software: the entrepreneur may choose either to develop the same features as the OS developers and gain the whole market thanks to its better user interface, or develop different features and concentrate on the users that have different tastes than the OS developers. It can in that case set a high price for its product even though it will benefit from lower network effects.

A discussion at the end of the paper will deal with the difference between BSD and GPL software. (LA)TEX's license is similar to the Berkeley Software Distribution ("BSD"), but the models are based on the assumption the OS license is the General Public License ("GPL").

1.1 Openness vs. efficiency and development objectives

The software engineering literature points to problems with coordination in OSS development. Herbsleb and Mockus (2003) show that distributed work faces difficulties as it slows down development, impedes changes to the software and leads to lesser communication between project participants than what could be achieved when development is done in the same place and under the same authority as happens in proprietary software development. Yamauchi, Yokozawa, Shinohara, and Ishida (2000) underlines limitations of strictly electronic forms of communication due to the lack of spontaneous conversation and the inability to share complex and ambiguous messages. On the other hand, OSS advocates point to the need in OSS development to back up any claim and projects with evidences as no argument of authority can be made. The need to state one's ideas in clear terms helps in improving them, and any development is archived for all to see (CVS archives), which allows anybody to follow and improve the work of others. It is therefore not clear coordination is always better in proprietary software development.

In the case of TEX, the organization went through four phases:

In the first phase, development was heavily centralized around D.E. Knuth at Stanford University, who set the goals for developers working with him and made suggestions for work by independent developers. He also queried for suggestions by the software's projected users, notably the American Mathematical Society ("AMS").

In a second phase, as D.E. Knuth withdrew from active development, a number of packages were developed in a rather decentralized way, leading to many replications in the development of similar features. The program became more complex and the number of programs linked to it rose: programs to draw figures, to do indexes, nationalizations of the software to deal with non-American typesetting traditions and characters, programs to output TEX in various alternative formats (HTML, XML, pdf...), user interfaces, rewritings of the software in various programming languages, etc.

There were several competing distributions of TEX on the market, each maintained by different persons. Several macro packages were developed for TEX. The LATEX package, developed by Leslie Lamport, rapidly became the most widely adopted. This was a stage of relative anarchy where the 'best' projects were those that succeeded in attracting the most users and on which the rest finally coordinated.

In a third stage, the TEX Users Group worked to propose a common distribution (TEXLive) with a common set of standards (The TEX directory structure) a common repository (The Comprehensive TEX Archive Network) and a central group of developers (The LATEX3 team).

The fourth and present stage is characterized by a return to decentralization. This is due to the combination of several factors: the Internet OSS community is now sufficiently organized to allow for the efficient coordination of projects on the Web, and there is not necessarily the need for a central forum where developers can meet face to face, such as the TEX users group. The software is increasingly international and responds to needs that the core group of developers did not predict, or that the TEX software cannot respond to without big changes. This means development is more and more specialized, or occurs outside of the original setting.

This shows a relative lack of efficiency of the organization in coordinating development; from the moment the original group of developers were not anymore working together in the same place, there were many example of parallel development of the same type of feature, either because developers didn't know each other, or because they disagreed on the objectives for development.

The difficulty in getting people to agree on objectives and pursuing them together is not wholly due to some inefficiency of the organization. In fact, some projects were ignored by the central organization and had to be pursued outside of its setting simply because they did not fit with the organization's objective. That difficulty in adopting new ideas and integrating new developers is due to network effects. Those are particularly important for typesetting software such as TEX: it was to be a document exchange standard, and this encouraged both better coordination of developers (they had to agree on any change in the software) and in making the software accessible to many users (the more people use a standard, the higher its value).

As the central organization was recognized as essential in maintaining a standard LATEX, it was difficult to initiate new projects without its approval. New users, with different needs than the ones originally intended for the program, would find it difficult to initiate projects in their own interests with the approval and support of the existing organization. The alternative was therefore between breaking away – but convincing many users that the new project is sufficiently interesting is almost impossible[2] – and integrating the core development team – but making the new goals understood and easy to implement takes time. This worked to slow development of innovative ideas in TEX, a problem that was exploited by some software firms: they were motivated to initiate those projects in the expectation of future profits drawn from users whose needs were not filled by the existing OSS.

Modelization: Coordination problems in open-source will be modeled the following way: There are two developers and two features to be developed. With probability α the two developers meet and agree on who will develop what feature. With probability $1 - \alpha$ they do no meet and they develop the feature that brings them the highest individual benefit. On the other hand, in proprietary software, an entrepreneur will hire developers and assign to them development tasks so that coordination problems will not be present.

The rigidity in development objectives of open-source organizations will be modeled the following way: There are two features to be developed and two kinds of consumers. One kind of consumers values the first feature higher than the second, while the second kind of consumers values the second feature higher than the first. The first kind of consumers controls the OS development process, so that the first feature will be developed OS instead of the second. On the other hand, in proprietary software, an entrepreneur will develop the feature that brings her highest profits.

1.2 Developing an interface for TEX and facilitating its use

Software is the addition of many features that are chosen and grouped together into a product, either by a software company or by an OSS organization. A distribution is the final product offered by an OSS organization to the common user of the software. On that distribution may be superposed an interface, which will be defined as any feature that facilitates the installation and use of the software.

Another way to see the "interface" in the case of enterprise software is the cost to pay an employee to maintain the software or the cost to obtain support from the software company that installed the software. Those costs can be very important, as about two-thirds of the cost of software is maintenance and support.

There are three stages in the practical availability of OSS to different groups of users: in a first stage, there are raw programs: those are usable only by sophisticated users. in a second stage, the programs are integrated in a distribution: this links many programs together and requires that each program conform to some guidelines. Not all programs are integrated a distribution, as not all developers are motivated to make the changes that are necessary for this to happen. In a third stage, a user interface is built on top of the distribution: this allows easier access to all the functionalities. This is also the stage where a quick way to install the distribution on mainstream operating systems is provided. In this paper, I make a further differentiation between providing an easy-to-install distribution, and providing a good user interface. TEX, like many OSS projects, was able to get to the stage of providing a complete easy-to-install distribution, but the OS user interface are not to the level of mainstream proprietary WYSIWYG text editors.

In the typology of Garvin (1984), OSS scores well on performance, feature, reliability and conformance, but is lacking in durability, serviceability, aesthetic and perceived quality. The first four are related to quality from the point of view of developers, while the last four are related to quality as perceived by users. Developers are able to compensate for the lack in the last four items, because they are expert users who can change the software when it becomes outdated, can fix its bugs and do not care so much for aesthetic as for ergonomics[3]. Nichols and Twidale (2003) is a

good review of usability in OSS and explain some of the technical difficulties OSS programmers face in developing an user-friendly software interface. Johnson (2002) gives another argument in part 5.3 of his article on why OSS may be less complete and user friendly than corresponding closed source applications. Open-source advocate argue that Linux has made improvements in its user-friendliness, and that OSS can be as user-friendly as proprietary software. While an individual user may not be able or willing to develop an user interface, individuals who are responsible in their organization for the installation, maintenance and usage of an OSS may be motivated to develop interfaces and write user manuals that they then can make available to the wider user community. Such efforts are indeed mentioned in the case study of TEX, when individual efforts in universities were then diffused on a word of mouth basis. Those OS efforts are however generally uncoordinated with the core development team, which doesn't have the same concern than those who develop interfaces.

In the case of TEX a good, fully-functional and easy to install TEX distribution was finally made available to the general user by the OSS organization, but its interface never was as aesthetically pleasing or ergonomically efficient as that of Microsoft's Word for example, that of competing proprietary typesetting software such as Quark, or of software based on TEX such as Microsoft Word.

TEX managed to be relatively accessible to the common user for three reasons:

The first is that it was developed in a domain that did not hold special interest for developers. This means that from the beginning on, the main motivation for its development was to make use of the software, not to signal programming ability. In fact, TEX came out of the lucky coincidence in one developer of the need for high quality mathematical typesetting and exceptional programming skills, combined with the fact Donald E. Knuth wanted his program to be accessible to the larger academic community so as to promote high quality typesetting.

The second is that TEX was meant to become a standard and this discouraged forking: developers were disciplined into producing software that respected TEX specifications. TEX license indeed prevented any program that did not pass some tests (TRIP/TRAP) devised by D.E. Knuth from calling itself TEX.

The third is that TEX's license allowed for proprietary implementations of TEX. Therefore, good interfaces, such as that provided by Scientific Workplace, were soon made available to the general public and contributed to making TEX more popular, even if not under an OS version.

As seen in part 1.2, the TEX Users' group was finally able to propose a standard easy to use version of TEX, which displaced proprietary versions. However, the success of the OSS organization was not total. Indeed, Microsoft's Word took many of the less sophisticated users of TEX, while publishers were attracted by Quark or Framemaker, which were competing typesetting tools with a much larger user base than TEX and could respond to the average non-mathematical publisher's needs.

Because of its lack of an interface, TEX was left either with those users who could not afford proprietary typesetting software (emerging countries, students), those people who needed its math-

ematical capabilities (academics) and those typesetters who valued the flexibility open source offers to devise their own specialized versions of TEX (Omega for multi-lingual typesetting was developed to typeset rare script and ancient documents with complex typesetting, ConTEXt was developed to produced interactive educational material).[4]

Modelization: The interface development problem in open-source will be modeled the following way: open-source developers will be assumed to derive no personal benefit from developing an interface, while the interface will have value ψ to all users. Developers and users will be assumed to derive benefit kx from the use of the software by proportion x of the population, with $k \geq 0$ (network effects). For k sufficiently high, a developer will therefore be motivated to develop an interface so as to increase the OSS's market share.

1.3 The alignment between users and developers' needs

The naïve version of how OSS and decentralized development manage to fulfill the needs of common software users is that any user can develop the feature she needs, so that any user's need is fulfilled with OSS. Of course, the mechanism is not so simple, as making changes in an OSS requires qualification that most users do not have. Franke and von Hippel (2002) and von Hippel and von Krogh (2003) underline a more complex system by studying the Apache open-source software and the objectives of its different classes of users. They show that developers' objective do not stray very far from the requirement of common users. In comparison with proprietary software development, von Hippel (1994) argues that OSS enjoys better access to sticky information about users' needs, those that are too expensive to determine in a one-to-one relationship between users and manufacturers.

In the case of TEX, another additional effect was at play, which consisted in consultation between the user base and developers. Gaudeul (2003b) cites examples of collaboration from the inception on between potential users of TEX and its developers. The AMS notably sent some representatives to Stanford to test Knuth's new Document Preparation System. The AMS financed the development of its own version of TEX and then of LATEX. Companies influenced the development of TEX through sponsorship of specific projects. PdfTEX to output TEX in pdf format was thus developed by Hàn Thế Thành partly under a scholarship from Adobe Corporation. The TEX users' group financed the development of a new version of TEX, NTS, that was meant to be more easily extendable than Knuth's program. Local Users' Group, those based in other countries than the US, also worked to fit TEX to their national needs.

Another mechanism by which users' needs and those of developers were aligned is simply that some features in TEX were inspired by previously existing and successful proprietary software (Bell Group's ROFF notably, or IBM's Script) and LATEX packages were inspired by Reid's Scribe. Finally, while proprietary versions of TEX were often the first to integrate some user-oriented features

(such as implementations for the PC by PCTEX or an user interface by Scientific Workplace or easier manipulation of a more diverse set of fonts by Y&Y), the OS developers frequently caught up quite quickly, although it is not clear how much of their development was inspired by proprietary advances.

Modelization: The alignment between developers and users' needs will be modeled this way: Developers assign value $\overline{v}$ to a first feature and value $\underline{v}$ to a second feature, while users assign value $x\overline{v}$ to the first feature and value $x\underline{v}$ to the second feature, with x distributed in the population according to a continuous distribution function f. This means that without even making any market research, an open-source developer will develop in priority the features that users most value.

1.4 Patterns in the competition between open-source and proprietary software

There is often competition between OS and proprietary software, as there is significant overlap between functions developed OS and those developed proprietary. Proprietary and OSS users' base are however significantly different (Feller and Fitzgerald (2000)): OS users are primarily expert users and early adopters, computer experts, small businesses and academic/research institutions. This paper gives some rationale for the difference in the software's user bases.

Few papers model competition between OS and proprietary software and its consequences on welfare, innovation and product design.

Schmidt and Schnitzer (2002) explain the coexistence of both type of software by resorting to an exogenous users' taste for each type of software. They underline how OSS reduces proprietary firms' motivation to innovate. In the present model, as proprietary software develops a better interface than OSS, there will be a differentiation between users with some of them buying the proprietary software for its interface even when the OSS provides equivalent features.

Baake and Wichmann (2004) analyze proprietary firms' choice to release some of their code OS and explain there is a balance to be found between the danger of facilitated entry by competitors and the benefit of decreased coding cost. Their model is useful to analyze the difference in strategies between firms basing their product on TEX such as Scientific Workplace, and firm such as Quark that wrote a software from scratch.

Khalak (2000) underlines the advantage proprietary firms enjoy in advertising their product, as they can cover that cost with future profits. Its point will be used later in the paper to explain how, when OS and proprietary software are equivalent and there are network effects, the proprietary software can gain the whole market and price its product so that no customer wants to switch to OS and lose the benefit of using the same software as everybody else.

Casadesus-Masanell and Ghemawat (2003) analyzes strategic reaction by proprietary software firms to the emergence of OSS and shows they can preserve their position on the market. They introduce demand-side learning effects in their model, whereby a dominant software remains dominant because it receives more suggestion for improvements or bug reports. This may be complemented

	Open-source	Proprietary
Based on TeX	User interfaces: TEXMacs and LyX. Distributions: NTS, ConTEXt and Omega, encouraged as successors to TEX. MiKTEX, an alternative to TEXLive	PCTEX and MicroTEX, early implementations for the PC. Scientific Word by McKichan Software, an user interface for Windows with added functionalities. Y&Y for fonts, TEXtures by Blue Sky Research for the Mac.
Developed independently	Abiword, Corel's Wordperfect which were proprietary software then released open-source. Various HTML/SGML/XML editors as XML replaces the TEX syntax as a mark-up language.	TROFF/NROFF for Unix by the Bell Labs, Scribe by IBM, Script were early competitors. Then came Adobe's Indesign, QuarkXPress, Framemakers for general typesetting, 3B2 for mathematical typesetting, Word for the common user.

Table 1: Typesetting software

by supply-side learning effects whereby you will invest more into a software that is popular. That paper is one of the few that introduce dynamic interaction between proprietary software firms and OS developers. Because demand-side effects introduce indirect network effects, the firm will price as a function of the market share advantage it has over OSS in a manner that is very similar to the present paper.

In the case of TEX, much has already been said in this paper about how both type of software competed over time. The table 1 p.10 recapitulates the various competitors of TEX along two dimensions, whether the software is open-source or proprietary, and whether it was based on TEX or not. The standard (LA)TEX open-source offering is itself very diverse. An official distribution (fpTeX for Windows, teTeX for Unix) based on the Web2c port of TEX into the C language, cohabits with other distributions maintained by individuals (MiKTEX, Bakoma, TEXnic) which all are based on TEX or LATEX but specialize in one area of typesetting. The chronology 2 p.37 charts the development of those software over time and Appendix E will be an helpful reference when discussing the dynamics of the development of TEX.

The previous parts have already underlined how the audience of each type of software differed or how OS and proprietary software borrowed features and ideas from each other.[5] The main raison d'être for proprietary versions of TEX was to provide an user interface and easy to install LATEX distribution. TEXLive, the OS distribution of TEX, and graphical user interfaces such as the shareware WinEdt or the freeware (not open source) WinShell thus spelled the end of many proprietary versions of TEX. Only those proprietary software that had diversified into features that the OS community was not able to develop, such as the WYSIWYG equation editor of Scientific Workplace and its integration of the Maple computation program, were able to survive.

The discussion of the difference between GPL and BSD OS software for what relates to the

pattern of competition between OS and proprietary software is relegated to the end of this paper. Even though proprietary software based on TEX could be marketed without revealing their source code, this did not spell in any way the end of that software's OS versions. The influence was more subtle, as proprietary versions broadened the appeal of that typesetting system and frequently acted as inducers into the use of the more sophisticated and flexible OS versions.

The rest of the paper is directed towards explaining the consequences and effects of three main factors in the competition between OS and proprietary software. The first is the lower efficiency of coordination in OSS development compared to proprietary firms, the second is how network effects influence the design of OSS such as to make it more accessible to the average user, the third deals with the difference in objectives between an OSS organization that cares only for its existing constituency, and a proprietary firm that wants to gain new customers and market shares. The models will allow to explain some characteristics of the user base of each type of software, and will show when a proprietary version of an OS software can be developed profitably.

The following model will serve as the basis for further modeling, and is thus detailed.

2 Description of a basic model

There are four types of players, users, open-source developers, an entrepreneur and an open-source software organization.

Users have the choice between a GPL licensed OS product which is sold at price 0 or a proprietary one which is sold at price P. The product is a software system that can include two features. Open-source developers value one feature at $\overline{v}$ and the other at $\underline{v}$. If both features are developed, the software's value to developers is $\overline{v} + \underline{v}$. Users order the value of each feature in the same way than developers, but the value they get from them is a fraction x of what developers get. That fraction x is distributed over the interval $[0, 1]$ according to the distribution function $f(x)$. Each user is characterized by its type x, that is her private knowledge. f is a logconcave distribution function, which means that the log of the cumulative distribution function of x is a concave function. This property is equivalent to the ratio of the density function to the c.d.f. being a monotone decreasing function. (Bagnoli and Bergstrom 1989). There is a mass n of potential users. For simplicity, one user who values a feature at fraction x of v will value the other feature at that same fraction of its value to developers.[6] x thus determines the proficiency with which a user may be able to use a software in a category, not the value she attributes to each of its features. Users face a cost ψ to make a productive use of the software, no matter how many functionalities it has.[7] That cost can be reduced to 0 if the software is provided with an interface.

2.1 Open-source development

There are two possible features that an open-source developer can implement, one with value $\overline{v}$ and the other with value $\underline{v}$. A developer can choose to develop the feature of value $\overline{v}$ or the feature of

value $\underline{v}$. Her cost to develop a feature takes a value $\overline{c}$ with probability λ and $\underline{c}$ with probability $1-\lambda$. A developer can also expend C to develop an user interface for the software program and thus reduce the users' learning and usage cost ψ to 0. However, the value of the interface is 0 to the developer as she already incurred learning and usage costs. She can develop only one feature or an interface.

One developer thus has the choice between developing a feature, an interface, or doing nothing and the payoff to a developer is therefore:

$$U_D = \max[v(a_D) - C(a_D), 0]$$

where a_D denotes the decision which feature to develop. From the above, $v(\text{first feature }) = \overline{v}$, $v(\text{second feature }) = \underline{v}$, $v(\text{interface }) = 0$, $C(\text{ develop a feature }) = \overline{c}$ or $\underline{c}$ and $C(\text{ develop the interface }) = C$.

Assumption 1

$$\overline{v} > \overline{c} > \underline{v} > \underline{c}$$

This means that with probability $1-\lambda$, it is not profitable for a developer to develop the lower valued feature, while it is always profitable to develop the first one.

OS development is sequential. In a first period, a first developer chooses which feature to develop. In a second period, a second developer comes. An OS developer develops a feature if it is not available and the cost of developing is lower than the value of the feature. However, in the second period, the developer is aware with probability α only of the work of the first developer.

2.2 Proprietary development

An entrepreneur has access to a pool of developers. A developer will develop a feature or the interface for a wage w. The entrepreneur doesn't get any value from using the software, but can sell it on the market and get its market value. Its market value is determined by the price P at which it sells the software multiplied by the number of users who buy it at that price. The entrepreneur chooses which features to develop, and whether to develop an interface. The entrepreneur is the one who decides which action a developer is going to take, and must hire one developer for every function she wants to see developed.

When choosing her price, the entrepreneur considers the following profit function:

$$U_E = Pn \int_x^1 f(z)dz - W$$
$$\text{with } x \text{ s.t. } u(x, \text{Prop}) \geq u(x, OS)$$
$$u(x, \text{Prop}) \geq 0$$

and W depending on the number of developers that were hired. Since a developer can develop only one feature, W will also depend on the number of features that were developed. $u(x,\text{Prop})$ denotes the utility of consumer of type x from consuming the proprietary product. The constraints ensure that the user indeed prefers the proprietary product to the open-source one, and secondly, that is rational for her to buy it.

2.3 Consumers' utility functions and decisions

After developers have made their development decisions, the open-source software product, which may include one or two features and an interface, is put on the market. An alternative, proprietary software product may also be available depending on the decision of the entrepreneur. Users choose which software to use (proprietary or OS) by comparing the value of the proprietary software (sum of the value of its features minus set-up costs minus price) to the value of the OS software.

The payoff to an user of type x who buys software of value v at price p is:

$$u(x) = xv - p - \psi$$

where

$$\begin{aligned} x &\sim f[0,1], \text{ uniform} \\ v &\in \{\overline{v}, \underline{v}, \overline{v} + \underline{v}\} \\ (p, \psi) &\in \mathbb{R}^2 \end{aligned}$$

and $\psi = 0$ if the product includes an interface.

The payoff to a developer who buys software of value v at price p is

$$u_D = v - p$$

i.e. $x = 1$ and $\psi = 0$ as explained previously.

2.4 Chronology of development

The chronology of development is the following:

- OS development happens first. It is sequential. In a first period, a first developer chooses which feature to develop. In a second period, a second developer comes. An OS developer develops a feature if it is not available and the cost of developing is lower than the value of the feature. However, in the second period, the developer is aware with probability α only of the work of the first developer.

It is always unprofitable to develop an interface, because that doesn't bring any value to a developer, and costs C to develop. It will therefore not be developed. It may be unprofitable to develop the lowest valued feature, given the assumptions on the cost of development and the value of the features. It is however always valuable to develop the highest valued feature. This means that with probability one, a first developer develops the highest valued function. Then, with probability α, a second developer learns about her work and develops the second most valued feature, provided it is valuable for her to do so. With probability $1-\alpha$, she doesn't learn about the work of the first developer and she develops the most valued feature as well. When OS development is finished, it is made available to all.

- The entrepreneur learns what features are available open-source but cannot integrate them in its product. She decides whether to develop a program, which features it will include and whether it will incorporate an interface. She then hires as many developers as features she wants to develop, pays wages and gets the features developed. When the product is finished, she sets its price P, with P set such as to maximize expected profits.

- Consumers learn P and the features that are included in both products and choose which product to buy and/or use.

Assumption 2

$$\overline{v}x^* \geq \psi$$

with $x^* = \max_x x(1-F(x))$

This means that the value of the interface is not too high. This will simplify the search for an equilibrium.

2.5 Analysis

There is a trade-off between quality (number of features that are included) and ease of use of the software (how convenient is the interface). The functionality and the interface aspects of the software are different because they bring different payoffs to users and to developers. OSS is limited by the fact developers are motivated by their own use of the software and not the use others may do of it (this translates in the interface not being developed).

The open-source and proprietary software will compete based on price and functionality. If the OS developers didn't coordinate their work and both developed the same function, and if it was profitable for the entrepreneur to develop both functions, then the OSS's functionality may be less than that of the proprietary software. The price of the proprietary software will have to be incentive-compatible for the users as they have the option to use the open-source software instead. The proprietary software is more efficient in terms of development cost (no duplication of effort, better coordination) but may produces less consumer welfare as its price excludes some users from the market.

Proposition 1 (Characterization of equilibrium: number of features, prices, market shares.) *Depending on w and ψ, 3 types of equilibria may emerge: either the open-source software is used by nobody, or it is used by low types users, or it is used by high type users.*

Proof. In appendix A ■

If development costs (w) and the value of the interface (ψ) are high, then the proprietary software may include less features than the OSS but will provide an interface. In that case, a full-featured OSS may compete with a proprietary software that includes less features but an interface. The OSS software will serve only higher value users, while the proprietary software will serve those customers who have a lower value for the software's features. In all other cases, the proprietary software serves higher value users. In those cases, the proprietary software will serve the same customers as if it was a monopoly except when its only advantage over the open source software is in its interface. In that later case, it will serve more consumers than if it was a monopoly and no consumers will use the OSS.

The proprietary software may include only the interface on top of the first function because the interface's value is the same for all, while the second function's value can be very low for some users. Proprietary software can be sold only as long as it adds features compared to the OSS. Both types of software do not compete on the basis of the most valuable, basic functions, but on the basis of the smaller, less valuable features which may not be valuable for an individual developer to work on, but will be developed by an entrepreneur because they provide a comparative advantage on the market. The proprietary software developer may decide not to develop both features, and instead develop only the first feature plus the interface. In that case, it will either keep its monopoly share of customers (in the case where the OSS has got only the first feature) or it will serve the lower end of the market (when the OSS is full featured).

We therefore have three cases: either sophisticated users use the OSS, or it is low-value users who do, or the OSS is not used by anybody. This model shows there are no cases where the OSS serves both low and high value users. An OS software's public is therefore either those users who have a high valuation for the software (Expert and commercial users) or those who have low valuation for the product (Third World, students). The second case is the one that has received the most attention but this model shows there is a special case when proprietary software may be less powerful but easier to use than OSS software and OSS users who have high value for the software's features forego the ease of use of the proprietary version for the additional capabilities of the OS version. That case also corresponds to some claims by OSS proponents, who point out that OSS may be better adapted to sophisticated users. Finally, it is interesting to note that in this model, whenever the only contribution from the proprietary software is to add an interface on the OSS product, then the proprietary software will get the whole market.

Those results can be illustrated with the TEX case study: there is a high demand for TEX in less developed, emerging or post-communist countries (New users' organization have sprung up in Hungary and China, for example). This corresponds to the case where OSS is used by 'low value

users'. There also is an use of TEX in high-end publishing, where other software are too limited even though they are readier to use. This corresponds to the case where OSS is used by high value users. Proprietary interface based on TEX also did take a lot of users who would have used the OSS otherwise, and left TEX with only developers as its public during the early development period. This corresponds to the case where nobody uses the OSS.

The process by which TEX managed to get back some of those users by developing better interface and better integration will be explained in the part about network effects; this process may have been due to the fear that the software would fall into oblivion if it did not manage to get some users back.

Proposition 2 (The impact of OSS on consumer welfare.) *Compared to a situation where OSS wouldn't exist, OSS unambiguously increases consumer welfare as long as it doesn't discourage the development of a full featured proprietary software with an interface.*

OSS decreases the price of the proprietary software but this doesn't necessarily increase the number of users who have access to it. The presence of an OSS never allows to attain the social optimum from the point of view of consumers as an interface will never be developed open-source, so that its zero price doesn't mean the whole market is served.

Proof. In appendix B ■

From proposition 2, there are many cases to consider when evaluating the effect of OSS on consumer welfare.

1. When both types of software are developed, and the proprietary software includes both features and the interface, the consumer welfare is higher than if there was only one type of software (proprietary). Indeed, there is a range of consumers who were not served before and who will be able to use the OSS, while the price of the proprietary software will be lower than in the monopoly case.

2. When both types of software are developed, but both include only one feature and the only advantage of the proprietary software over the OS one is the interface, then the consumer welfare may be lowered compared to the monopoly case even though the price of the proprietary software is lowered and more customers have access to it. This is because the second feature is not developed.

3. If the presence of the open-source alternative discourages proprietary development of the interface, then the result is ambiguous as the price of the proprietary software is lowered and a portion of consumers who were not served now have access to the OSS, but all consumers must now incur higher set up costs.

4. If the presence of an open-source alternative discourages proprietary software development altogether, then the consumer welfare may be lowered: more customers have access to the

software, but since it is OS, it doesn't include an interface and it may also include less features than the proprietary version.

The existence of OSS potentially decreases consumer welfare, in case the proprietary software is not developed and one of the features is not developed under OS conditions. If the OS organization is effective (high α, so that the probability developers are aware of each other's work is high) and the probability λ there is high cost of development is average, then there are lower chances the proprietary software is developed. A very high λ favors proprietary software, because it means there is a high probability that the OSS will include only one functionality, and a very low λ also favors proprietary software because it means development costs will be low (assuming the wage w is related to the average cost of development Ec). Consumer welfare is improved the more likely it is the proprietary software's only advantage is in its interface as it will have to price at ψ and sell to more customers than otherwise.

The case where the proprietary software developed both features plus the interface and the OSS also developed both features is therefore the most efficient. The more efficient the OSS organization is, the more likely the proprietary firm sets a lower price and serves more users. OSS therefore has an effect on proprietary software production beyond reducing its profit as it also potentially forces it to serve more consumers; in that case, the effect on consumer welfare is unambiguously positive as long as the proprietary software develops both features and the interface.

The proprietary firm enjoys an advantage over the OS organization in that it is the only one to be able to develop an interface. Due to incentive constraints, the OS developers are not motivated to develop it, and this allows the proprietary firm to maintain the appearance of monopoly: either it serves the same customers as if it was a monopoly, and leaves the lower end to OSS, or it serves all the market.

There is a part of the market which still isn't served, the portion of those people who have a low value for the software, so that they don't buy the proprietary one, and don't use the OS software either because set-up costs are too high. This means OSS doesn't allow to reach the social optimum.

Public policy The public policy question about open-source software often concerns States that have no software industry to care for, but have to decide whether they should encourage the use of OSS alternatives to software bought from foreign companies. In that calculation, they therefore only consider consumer welfare. OSS has the advantage of being free, but may have less features and be more difficult to use than proprietary software. However, its existence encourages the proprietary software to lower its price.

For a country with an important software industry, and which therefore considers global welfare in its calculations, it may be a good idea to encourage OSS development as long as it doesn't discourage the development of a full featured proprietary software and the costs of development are not too high. The costs of development must not be too high because else, it is not advisable to encourage double development (OS and proprietary) of the same types of features. The effect on

consumer and global welfare is even more ambiguous if OSS discourages proprietary development of a feature or of the interface, but it can still be positive in that case.

3 Network effects:

In this part, we consider the impact of network effects on the pricing and design decision of the entrepreneur and of open-source developers. For simplicity, assume there are no set-up costs so that the interface is never developed.[8]

The consideration of network effects is important for three reasons: 1) they arise naturally from the way open-source software is developed, 2) open-source developers attach a lot of importance to them, and 3) open-source and proprietary software are seldom used together, which means that the networks they draw on are very differentiated.

1. Open-source software draws its developers from its users. Mockus, Fielding, and Herbsleb (2000) and Mockus, Fielding, and Herbsleb (2002) and their study of the contributors to the Apache project, Ghosh and Prakash (2000) and the Orbiten free software survey, have outlined three types of way by which users contribute to the quality of the OS software: a first, small group of developers develops most of the code, a second group contributes fixes and suggestions for changes, and the last, largest group report bugs.

   The incremental contribution of each individual in those groups is a decreasing function in time as the main problems get solved in priority, but each contribute positively to the quality of the software. There is also a process of conversions of users into developers that is at work; users go up the OS learning curve and are frequently able to contribute to its development after some time using it. While that effect is not present to such an extent in proprietary software, it also exists as users report problems and the firms are more motivated to develop the software the higher the number of potential users there are. Of course, for many software, there also are direct network effects, as the value of a software increases the higher the number of people you can exchange data and document with it. In the case of TEX and of other typesetting software, standards for document processing and output are very important. The TEX community gives a lot of importance to the preservation of TEX as a standard for mathematical typesetting, thus recognizing the importance of network effects.

2. The OS developers frequently have a missionary attitude to OSS, and often define themselves in opposition to proprietary alternatives. They try not only to develop good software that fits their needs, but they also try to convert other developers and users to OS. Stallman's FSF was founded precisely to encourage such aims, and he hoped "to supply, eventually, everything useful that normally comes with a Unix system" (Stallman 1983). The project was aimed at providing an alternative to a proprietary software. Linux was generated as an alternative to Andrew Tannenbaum's Minix, which was too expensive for Linus Torvalds.[9]

Linux users are frequently seen as anti-Microsoft, and there is generally a high level of distrust between the OS community and proprietary software firms. A great part of that defiance is due to the perceived need by OS users to establish a large user base for OSS to become more than a niche market, and for its development to be dynamic. OSS developers are therefore conscious of network effects, and take it into account when making their development decisions. Indeed, some projects relating to Linux, such as Gnome, are primarily directed towards producing a rival to MS's Windows. They want to make an user interface to Linux such as to make it as easy to use as Windows, and thus distract some of its users.

3. OSS and proprietary software are seldom used together by the same users; there is little dual use. This is not only because OSS software primarily works on Unix operating systems, while proprietary software often works on top of Windows, and most users use only one operating systems for most of their needs. This is also because they use different standards, and it is often difficult for a Windows user to open a document created using OSS, and vice-versa. There is a considerable difference too in the type of users both software attract, and this is not only due to those users having different needs and thus using different software for different functions. Indeed, there is almost always an OS alternative to proprietary software.

The main point of this part is to show network effects introduce incentives for OS developers to develop features that are less valuable to themselves, but of value to the market. The need for OSS to reach users (who may become developers, or simply act as testers), is an incentive for OSS developers to compete for market shares with proprietary software. This part therefore introduces strategic elements in the behavior of OSS developers, who previously were merely internally motivated.

The payoff to an user of type x who buys a software of value v at price p and which is used by n users is now:

$$u(x) = xv - p + kn$$

where

$$\begin{aligned} x &\sim f[0,1], \text{ uniform} \\ v &\in \{\overline{v}, \underline{v}, \overline{v} + \underline{v}\} \\ (n, p, k) &\in \mathbb{R}^3 \end{aligned}$$

There are two software, one proprietary and one open-source, and the two products are supposed to be incompatible: n is the number of users of the software the user uses, not the total number of people who use one or the other software.

Proposition 3 *Network effects have two contradictory results: As long as the OSS doesn't include both features, the higher the network effects the higher the proprietary software's market share, but*

the probability the OSS includes both features increases with k and when the OSS includes the two features the proprietary software is out of the market.

Proof. In appendix C ■

There are two contradictory effects at play: if network effects are very important, the proprietary software will want to get all the market but this gives OS developers more incentive to develop both features, and in that case the proprietary software's market share may be lowered to 0. Proprietary software developers are thus in an awkward position as they cannot commit to leave a part of the market to the OSS, which encourages OSS developers to compete more strongly with them, and thus potentially gain the whole market.

The higher the network effects, the higher the number of consumers who will buy the full featured proprietary software whenever that one is developed, but also, the higher the probability that a fully featured OSS is developed. The resulting market condition are therefore more clear cut: the higher the network effects, the more people use the proprietary software if it is developed. However, high network effects mean the probability it will not be developed because a full-featured open-source alternative will have been developed to counter it pre-emptively.

Proprietary profits may or may not increase with network effects, so that as k increases, proprietary development may be discouraged quite apart from the fact a free alternative emerges.

4 Organizational objectives and competition

Two factors in the competition between OS and proprietary development were underlined in the two previous parts: The better coordination in proprietary organizations, as well as the way features developed OS or proprietary are chosen (based on cost of development in OS, on profit potentials in a proprietary model). Those choices processes were shown not to be equivalent.

Other aspects were not taken into account, especially some which favor the OS software organization, notably a better fit between the software functionality that are developed OS and the way the software is used by the average user in the general users' community. There is indeed a tendency for proprietary software to become bloated as development is directed toward providing new functionalities so as to attract new segments of the users' population, at the expense of improving existing functionalities that are of use to the broader users' population.

This part deals with the difference in features choices between OSS organizations and proprietary software firms. The case study outlined the difference between proprietary firms that want to attract new users to their product by including features that they value the most, and an OSS organization that, at least in the case of TEX, cares mostly about existing users and tries to improve the existing software, not find new domains of application or increasing its market share. The issue is in fact rather muddled, as OSS advocates say that any new user with new requirement can adapt

the software to its own need, so that OSS would be more efficient than proprietary software at satisfying the fringe user. Proprietary software, on the other hand, would be unable to be as versatile in its application because of two effects: the user-interface which is difficult to change and adapt to integrate new functions, and the closeness of the development process, which shields developers from new influences and users' requirement.

However, in the case of TEX new radical projects did have difficulties getting off the ground, and this was partly due to organizational inertia (but also to the difficulty for an OSS organization with very limited funds to undertake radical restructuring of a mature program). That inertia was related to the fact those improvements concerned only a fringe of the user base, and to high network effects that dissuaded those who potentially would have been interested in the new projects from leaving the old project and forfeiting the benefit of an established base of users with whom to exchange documents with. On the other hand, proprietary software based on TEX did frequently propose new and original features that were in demand by users outside of the realm of interest of existing TEX users (most notably, an user-interface, but also original and easy to manipulate fonts sets (Y&Y), or a computation software (MuPAD in Scientific Workplace)). The contrast is not clear cut, since the OS TEX organization did come to integrate new font sets, a complete distribution and also pdf format output. But there is clearly a difference in emphasis and in financial means that resulted in those improvements coming later into standard OS use than in proprietary software based on TEX.[10]

Let us therefore look at a variation on the model. The difference with the preceding models is that there are two types of consumers and two software products that may or may not have the same functionalities. As before, there are two functionalities, but they do not have the same value depending on whether you belong to one type of consumer or the other. As in the preceding part, there are network effects and the two products are incompatible. Unlike the previous parts, only one feature may be developed: the OS community and the entrepreneur must simply decide which feature to develop for their respective product.

There are two kinds of users of type $t =$ old ('O') and new ('N'), in proportion s and $1 - s$ respectively, who value two features, $f = 1$ and 2 differently. Old users value feature 1 at v_{1O}, and feature 2 at v_{2O}, new users value feature 1 at v_{1N} and feature 2 at v_{2N}.

Assume that

$$v_{1O} \geq v_{2O}$$
$$v_{1N} \leq v_{2N}$$

so that the old users would prefer to see the first feature developed, while new users would prefer to see the second one developed.

An OS organization and a firm decide at the same time which feature to develop. They can develop only one feature due to time and money constraints. The open-source organization is composed of members of the old user-base, and will therefore decide to develop the feature that

benefits that part of the users' community, while the proprietary firm will develop the feature that brings it highest profits. It sets its price P such that it serves a portion $1 - x_O$ of the old user base and portion $1 - x_N$ of the new users. The rest are served by the OSS, whatever feature that one develops.

There are network effects and the two products are incompatible, so that the payoff to an user of type x belonging to section $t \in (O, N)$ of the user base who buys a proprietary software with feature $f \in (1, 2)$ at price P is:

$$u_P(x, f, t) = xv_{ft} - P + k[(1 - s)(1 - x_N) + s(1 - x_O)]$$

while the payoff of using an open-source software with the same feature is:

$$u_{OS}(x, f, t) = xv_{ft} + k[(1 - s)x_N + sx_O]$$

where

$$\begin{aligned} x &\sim f[0, 1], \text{ uniform} \\ t &\in \{O, N\} \\ P &\in \mathbb{R} \\ v_{ft} &\in \{v_{1O}, v_{2O}, v_{1N}, v_{2N}\} \\ (x_N, x_O, s) &\in [0, 1]^3 \end{aligned}$$

Proposition 4 *The proprietary software will decide either to serve only the new user base by developing the feature they most value, or will serve the whole market by developing the feature that is most valued by the old user base. It is only for* $k \leq \frac{v_{2N} - v_{1N}}{1-s} \frac{3-s-2\sqrt{s(3-2s)}}{9}$ *that the second feature will be developed.*

Proof. In appendix D ■

The fact that for k low relative to $v_{2N} - v_{1N}$, the entrepreneur will prefer to develop the second feature and concentrate on the fringe is understandable: having a large market share is not worth so much (network effects are small) while the fringe's relative value for the new feature is high ($v_{2N} - v_{1N}$ is high). For k high relative to $v_{2N} - v_{1N}$, the opposite will occur: the firm will want to develop the first feature and sell to the whole market at price k.

For k medium and s low, the firm could prefer screening customers based on their type rather than exploiting network effects to the maximal extent, but that case doesn't happen with the specifications of the model (linear network effects, uniform distribution for consumers' types).

(Graph 1 p. 23)

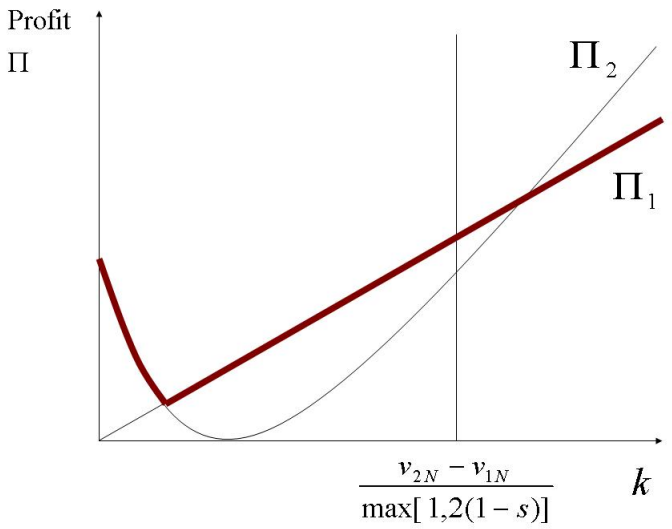

Figure 1: Profit as a function of k depending on the feature developed.

The graph above represents profit as a function of k (bold line). If the first feature is developed, profit is a linear function of k (Π_1). If the second is developed, it is a convex function of k (Π_2). Therefore, there should be a level for k high where it becomes more profitable to develop the second feature again. However, for such k, it becomes impossible to convince new customers to buy the new feature, because they prefer joining the old users and use the OS product with the first feature. This is because k is so high that the benefit of everyone using the same software outweigh the benefit of using a software that better fits your aims, and it becomes impossible to sell the product with the new feature. This explains why there is no level of k high where the firm prefers screening customers from the new user base rather than sell to all and come into direct competition with the OSS. There is also no level of k medium high, where the profit with the new feature is higher than with the old feature, and consumers can be tempted into buying the new feature. This is because the limit above which the consumers' always prefer to all use the same feature ($\frac{v_{2N}-v_{1N}}{\max[1,2(1-s)]}$) is lower than the limit above which profit with the second feature is higher than profit with the first.

This could be put into relation with the choice of feature development by proprietary software companies that built their product on TEX. It is possible from this model to infer from k quite high in the case of TEX that the optimal strategy for the firms is to develop the same kind of features as what OS developers would want. Of course, this is valid only if, as was assumed here, the proprietary firm is able to impose sufficient costs of switching from its product to the OS one, so that the two products can be considered as incompatible: an OS user would not benefit from the presence of the proprietary users and vice versa. In the case of TEX, this is indeed the strategy that was followed to some extent, with Scientific Workplace for example using some proprietary

functionalities that make it difficult to transcribe documents generated with it into common LaTeX format, or Y&Y using proprietary fonts. Changing that assumption of no compatibility will result in lowering profits when both systems produce the same feature, but doesn't change the main insights of the proposition.

This part therefore explains why the features that were developed in a proprietary way, apart from the interface, where soon included into the OS version too (more fonts, better integration of functionalities, capacity to output pdf files, etc...). They were functionalities that the original users needed too. The only advantage of the proprietary firm was therefore in being able to propose an interface and offer new features to the general users population faster than the OS organization (see note 12). In the longer term, as the product becomes complete and users more expert in the use of the software and don't need the interface anymore, they ultimately all end up with the OS version. This is what seems to have happened in the case of TeX: firms that based their product uniquely on TeX almost disappeared from the market, as their only advantage was in their interface which was valuable to smaller and smaller cohorts of new users. The only survivors were those that diversified and included non-TeX programs into their package.

5 Discussion of the difference GPL vs. BSD

In the present paper, the GPL license was assumed to be the only one available. In the GPL, the entrepreneur cannot use the features that were developed open-source. If they are under the BSD, then the entrepreneur could use them and integrate them into her proprietary product. That distinction can be seen in practice, as GPL open-source software was developed as an open-source equivalent of existing proprietary software (GNU/Linux as a replacement for Unix, Gnome as a replacement for Windows) while there are many proprietary software that use BSD open-source elements in the background or as an interface to open-source software (Scientific Workplace for TeX). However, there are proprietary distribution of GPL open-source software (Red Hat for Linux) and it is possible to build proprietary features on top of GPL software so that the difference between the two shouldn't be overemphasized.

In the terms of the first model with the interface, assuming the license is BSD instead of GPL changes two results: the first is that the case where the proprietary software has less features than the OSS disappears. There is therefore no case where the proprietary software serves low value users and a BSD OSS serves high value ones. The second change concerns welfare: it is always higher with the BSD, because there is an economy in development costs. What is developed OS can be exploited by a proprietary firm.

Another difference between the GPL and the BSD case is not taken up by the model: With the BSD we can potentially have three types of software, one 'pure proprietary' which features are developed independently, and then sold on the market, one 'mixed proprietary' which borrows features from the open-source software and builds on it to sell on the market, and one pure open-

source software. However, the mixed-proprietary software will always enjoy an advantage over the pure proprietary software, because it will in any case save on the cost of developing the highest valued feature. The BSD case is therefore better from the point of view of welfare than the GPL case because the proprietary firm economizes on development cost, and its price will be equal or lower than in the equivalent situation when the software is BSD. Consumer welfare is therefore also improved.

However, and this is where the model finds its limit, mixed proprietary software is more vulnerable to competition from open-source software than pure proprietary software. Indeed, because it is at least partially compatible with OS software, it is easier for its users to switch to the pure OS version as soon as that one offers greater ease of use. A model could therefore be developed where proprietary firms balance reduced development cost with lower possibilities to exploit network effects to their exclusive advantage.

In a companion paper, Gaudeul (2004) explains the influence of software's license terms on its development.

6 Conclusion

This is one of the first paper looking at competition between two radically different software development techniques, open-source and proprietary. The originality is in looking at how that coexistence influences product design, both from the point of view of OS developers and from the point of view of software firms. Three aspects were studied: how proprietary software companies can maintain themselves in the face of OS competition by offering a better user interface, how network effects encourage OS developers to develop more features, and how proprietary firms may want to differentiate by developing different features than OS ones.

The paper explains many findings from the (LA)TEX case study, notably why proprietary software developers basing their product on TEX attach so much importance to the interface as a way to survive faced with the OS version. It also explains how the existence of a proprietary alternative led TEX developers to devise ways to better take into account the interests of the general user population, and not, as is often assumed, exclusively their own. This led to the creation of an influential and efficient network of users' organizations that greatly contributed to the continued dynamism in TEX development, and to its democratization and wide use. Finally, the paper explains proprietary software may sometime be more responsive to new users' needs than OSS. The two strategies used by proprietary firms were therefore to design better interface and develop features for users with special needs who were unable to develop them themselves or to use the OS solution.

7 Extensions and other perspectives

An obvious extension would be to combine the three partial models we have into one. This would notably allow to better understand how network effects influence the software design, as in the first model with network effects, high network effects could lead either to all the market going to the proprietary software or to the OSS, while in the second, high network effects lead to the whole market going to the proprietary software.

The possibility that some segment of developers do not value each feature the same way than others was introduced, without acknowledging that this may increase the number of features that will be developed OS: a feature may have a lower value on average for the market, but high value to some, who will therefore be motivated to develop it even if development costs are high. There would therefore be higher probabilities that the OSS is full-featured.

On the other hand, there may also be a difference in each type of software's target consumers: Sophisticated users can with some effort make the few adjustments necessary in open-source software so as to use of it efficiently, while unsophisticated users need all features pre-installed to make use of the software. The first type of consumers would decide to use the open-source software OSS based on a cost comparison with the proprietary software, while the second type would never be able to use OSS. This would reduce the impact of OS on proprietary software production.

A dynamic model would allow to take into account the differences in pace of development of OSS vs. proprietary software, as this is a matter of much discussion; OSS would always be one step ahead in terms of the development of basic, most valuable functions, while proprietary software would elaborate on those functions so as to make them marketable to the general public. The interface would have to be constantly updated to integrate new functions, which means there would be a delay in the integration of new functions.

As explained in another part of the paper, there is a symbiotic relationship between OS and proprietary software, as proprietary software based on OSS acts as a first step towards the use of the more sophisticated OS versions of the software. On the other hand, OSS development provides the basics upon which the proprietary software can develop. The relation is symbiotic because one is not always beneficial to the other. Switching from one version of the software to another could therefore be modeled as consumers gain expertise in its use (encouraging them to use the OS version) and as the proprietary software's interface becomes more sophisticated (encouraging consumers to use the proprietary version). Proprietary software based on open-source must also be distinguished from proprietary software that is developed independently: the model doesn't allow to understand the distinction other than in trivial terms.

References

BAAKE, P., AND T. WICHMANN (2004): "Open source software, competition and potential entry," Berlecon Research Papers.

BAGNOLI, M., AND T. BERGSTROM (1989): "Log-concave probability and its applications," Unpublished manuscript.

CASADESUS-MASANELL, R., AND P. GHEMAWAT (2003): "Dynamic mixed duopoly: A model motivated by Linux vs. Windows," Harvard Business School Working Paper #04-012.

CRÉMER, J., AND A. GAUDEUL (2004): "Quelques éléments d'économie du logiciel libre," *Réseaux, forthcoming*.

FELLER, J., AND B. FITZGERALD (2000): "A framework analysis of the open source software development paradigm," 21st international conference on information systems, Brisbane, Australia.

FRANKE, N., AND E. VON HIPPEL (2002): "Satisfying Heterogeneous User Needs via Innovation Toolkits: The case of Apache Security Software," MIT Sloan School of Management WP #4341-02.

GARVIN, D. (1984): "What does "Product Quality" really mean?," *Sloan Management Review*, pp. 25–43.

GAUDEUL, A. (2003a): "The (LA)TEX project: A case study of open-source software," *TUGBoat*, 24, 66–79.

——— (2003b): "The (LA)TEX project: A case study of open-source software," http://agaudeul.free.fr.

——— (2004): "Open source software development patterns and license terms," Working Paper.

GHOSH, R. A., AND V. V. PRAKASH (2000): "The Orbiten Free Software Survey," *First Monday*, 5(7).

HERBSLEB, J., AND A. MOCKUS (2003): "An Empirical Study of Speed and Communication in Globally-Distributed Software Development," *IEEE Transactions on Software Engineering*, 29(6).

JOHNSON, J. (2002): "Open Source Software: Private Provision of a Public Good," *Journal of Economics & Management Strategy*, 11(4), 637–662.

KATZ, M., AND C. SHAPIRO (1999): "Antitrust in Software Markets," in *Competition, Innovation and the Microsoft Monopoly: Antitrust in the Digital Marketplace*, ed. by J. Eisenach, and T. Lenard. Boston: Kluwer Academic Publishers.

KHALAK, A. (2000): "Economic model for impact of open source software," Massachusetts Institute of Technology.

LERNER, J., AND J. TIROLE (2004): "The economics of technology sharing: open source and beyond," Working Paper.

MOCKUS, A., R. FIELDING, AND J. HERBSLEB (2000): "A case study of open source software: The Apache Server," in *Proc. 22nd. International Conference on Software Engineering, Limerick, IR*, pp. 263–272.

MOCKUS, A., R. FIELDING, AND J. HERBSLEB (2002): "Two case studies of open source software development: Apache and Mozilla," Working Paper.

NICHOLS, D., AND M. TWIDALE (2003): "The usability of open source software," *First Monday*, 8(1).

SCHMIDT, K., AND M. SCHNITZER (2002): "Public Subsidies for Open-Source? Some Economic Policy Issues of the Software Market," CEPR Discussion Paper.

STALLMAN, R. (1983): "Original announcement of the GNU Project," http://www.gnu.org/gnu/initial-announcement.html.

VON HIPPEL, E. (1994): "Sticky Information" and the Locus of Problem Solving: Implications for Innovation," *Management Science*, 40(4), 429–439.

VON HIPPEL, E., AND G. VON KROGH (2003): "Open Source Software and the "Private-Collective" Innovation model: Issues for Organization Science," *Organization Science*, 14(2), 209–223.

YAMAUCHI, Y., M. YOKOZAWA, T. SHINOHARA, AND T. ISHIDA (2000): "Collaboration with lean media: how open source software succeeds," ACM 2000 conference on computer supported cooperative work.

Notes

[1]Gaudeul (2003b) is a long version of the case study, available on http://agaudeul.free.fr.

[2]This is the chicken and egg problem: if many users choose to use the new project, then others will do so too, but if none do, none will. In the absence of a coordination device, an individual will not use the new project in the belief others won't either. This is because if she is the only one to participate in the new project, and even if the project is very good, there is almost no chance she will be able to finish it.

[3]OSS arguably is more ergonomic to an experienced user than proprietary software with many 'intuitive' features.

[4]Note however that some TEXdevelopers want to contribute to the success of Linux in contesting the Microsoft desktop monopoly and are trying to develop Word-like interfaces for TEX. (LYX, XeMTEX, TEXMacs, ...) While those developers see improvements in TEX mainly as a way to increase the value of the Linux operating system, they contribute in getting some new users to TEX and prevent the number of users of TEX from becoming so low as to endanger its survival.

[5]Indeed, some innovation in TEX were sometime directly integrated in proprietary products, such as its hyphenation and justification algorithm, or quite simply, the idea that writers could deal with most typesetting tasks.

[6]This means that if both features are developed and the user is of type x, she will get value $x(\overline{v} + \underline{v})$ from using the software.

[7]That cost can be divided along two dimensions: a cost which depends on how much they will use the software (which is related to the parameter x), and a fixed cost to install it and learn how to use it. Proprietary software is frequently seen as easier to install, learn and use than OSS. This is attributed to a motivation problem for developers, who do not benefit as much as users from designing helps for installing and learning the software, since they already installed and learned it. However, there is a difference between the fixed, set-up costs of the software, and the continuing, usage costs of the software. Those last ones are incurred by developers and users alike, and probably even more so by developers because they are heavy users of the software. It could be argued that the cost of using an OSS is lower in the long term than for proprietary software, because OSS frequently provides more shortcuts and customization possibilities than proprietary software. This may be due to the incentives of developers to develop software that is easy to use, if not easy to learn. In the model, we neglect those usage costs and concentrate on fixed set-up costs.

[8]Alternatively, consider the second feature as a feature easing the use of the software and whose value is directly related to how valued the first feature is. See note 1 on the difference between usage cost and installation cost.

[9]For more on the Linux/Minix controversy, see: http://www.oreilly.com/catalog/opensources/book/appa.html. Note this comment by Peter MacDonald: "I feel compelled to comment on my reasons for switching from Minix to Linux. In order of importance they are:

1) Linux is free

2) Linux is evolving at a satisfactory clip (because new features are accepted into the distribution by Linus)."

[10]The process of development and democratization of OSS is three-fold:

Sophisticated users find ways to integrate new features (in the case of TEX, design new fonts, produce pdf files). Their needs usually prefigure future needs in the general users' population.

A standard solution emerges. Commercial developers, who in the case of TEX can be well integrated into the OS developers'population, take it up and integrate the new feature into their proprietary system.

The standard solution becomes available to those in the general user population that cannot or don't want to pay the commercial software companies. This is done through individual or users' group sponsored efforts.

A Proof of proposition 1

The entrepreneur maximizes profits subject to $u(x,\text{Prop}) \geq u(x, OS)$, and $u(x,\text{Prop}) \geq 0$. Define $\psi(prop)$ the set-up cost for the proprietary software, and $v(prop)$ the value of the proprietary software's features. The first constraint admits three cases: if $v(prop) > v(OS)$, then customers of type $x \geq x_2 = \frac{\psi(prop)+P-\psi}{v(prop)-v(OS)}$ buy the proprietary software. If $v(prop) = v(OS)$ then we must have $P \leq \psi - \psi(prop)$. If $v(prop) < v(OS)$ then customers of type $x \leq x_2$ buy the software. The second constraint says that customers of type $x \geq x_1$ will buy the software, with $x_1 = \frac{\psi(prop)+P}{v(prop)}$.

There are different cases, depending on which features the entrepreneur chose to develop. Obviously, the entrepreneur will have developed the highest valued feature in preference to the lowest valued one, since they both cost the same to develop. She will also have developed either the second feature, or the interface, or both, else she wouldn't be able to sell her software at a positive price to anybody (indeed, the open-source software will always include the highest valued function).

There are therefore three cases, either the entrepreneur only developed the second highest valued feature (Case A), or only the interface, (Case B) or she developed both (Case C).

Case A:

Suppose that the open-source software includes the highest valued feature only. This happens

with probability $1-\alpha+\alpha\lambda$. Indeed, with probability $1-\alpha$ the two developers were not aware of each other, and with probability α the second one was aware of the first one's work but faced too high a cost to develop the second feature. There is therefore an alternative of value $\overline{v}$ to the proprietary software and the entrepreneur can sell her software for a price P to maximize profits subject to the users' incentive compatibility constraint. The software is thus sold to a fraction $\pi(P) = \int_x^1 f(z)dz$ of the n potential users, s.t. $u(x,\text{Prop}) \geq u(x, OS)$, and $u(x,\text{Prop}) \geq 0$. Consumers value for the second feature is distributed according to $\underline{v}x$ with $x \sim f(x)$ a c.d.f. over $[0, 1]$.

The entrepreneur's profit function is therefore

$$U_E = Pn \int_x^1 f(z)dz - 2w$$

s.t.

$$\underline{v}x - P > 0$$

$$(\overline{v} + \underline{v})x - \psi - P > 0$$

Then x^* is such that $x^* = \frac{1-F(x^*)}{f(x^*)}$, s.t. $xf'(x) + 2f(x) > 0$ (Concavity of the profit function), and $P_A^{*1} = \min[\underline{v}x^*, (\overline{v} + \underline{v})x^* - \psi] = \underline{v}x^*$ by assumption (Notation: P_A^{*1} is the optimal price in case A when the OS product includes 1 functionality). The OSS serves consumers in the interval $[\frac{\psi}{\overline{v}}, x^*]$.

If the open-source software includes both functionality, then the entrepreneur cannot sell her software at a positive price, and $U_E < 0$. The proprietary software is therefore not developed and the OSS serves consumers in the interval $[\frac{\psi}{\overline{v}+\underline{v}}, 1]$.

Case B:

If the open source product includes only one feature, the entrepreneur maximizes $U_E = Pn \int_x^1 f(z)dz - 2w$ s.t. $\psi - P \geq 0$ and $\overline{v}x - P \geq 0$. Profit is thus either $\psi n \int_{\frac{\psi}{\overline{v}}}^1 f(z)dz - 2w$ or $\overline{v}x^* n \int_{x^*}^1 f(z)dz - 2w$ s.t. $\overline{v}x^* \leq \psi$. Since $\overline{v}x^* \geq \psi$ by assumption, profit is then $\psi n \int_{\frac{\psi}{\overline{v}}}^1 f(z)dz - 2w$ and $P_B^{*1} = \psi$. No consumer uses the OSS.

If the open-source software includes the two features, then she sells her software to any users s.t. $\psi - P \geq x\underline{v}$ and $\overline{v}x - P \geq 0$. She thus maximizes $Pn \int_{\frac{P}{\overline{v}}}^{\frac{\psi-P}{\underline{v}}} f(z)dz$ so that $P_B^{*2} = \frac{F(\frac{\psi-P_B^{*2}}{\underline{v}}) - F(\frac{P_B^{*2}}{\overline{v}})}{\frac{1}{\underline{v}} f(\frac{\psi-P_B^{*2}}{\underline{v}}) + \frac{1}{\overline{v}} f(\frac{P_B^{*2}}{\overline{v}})}$ and the entrepreneur sells to consumers in the range $[x_L, x_H]$ with $x_L = \frac{P_B^{*2}}{\overline{v}}$ and $x_H = \frac{\psi - P_B^{*2}}{\underline{v}}$. x_H will be more than x_L or $P_B^{*2} \leq \frac{\psi\overline{v}}{\overline{v}+\underline{v}} < \psi$. Also, we will have $u(x_L, OS) = (v+v)x_L - \psi \leq 0$: the OSS software will serve customers who have a high value for the software's features (interval $[x_H, 1]$) while proprietary software will serve those who have medium value for it (interval $[x_L, x_H]$) The proprietary software's profit is thus $P_B^{*2} n \int_{x_L}^{x_H} f(z)dz - 2w$.

Case C:

If the open-source product includes only one feature, then the proprietary software firm will sell to any user such that $P - \psi \leq x\underline{v}$ and $(\overline{v} + \underline{v})x - P \geq 0$. Her profit is therefore $\min[x^*\underline{v} + \psi, (\overline{v} + \underline{v})x^*]n \int_{x^*}^1 f(z)dz - 3w = (x^*\underline{v} + \psi)n \int_{x^*}^1 f(z)dz - 3w$ by assumption and the price is

$P_C^{*1} = P_A^{*1} + \psi$. The OSS serves consumers in the interval $[\frac{\psi}{\overline{v}}, x^*]$.

If the open-source product includes two features, then the entrepreneur can sell her product at a price P such that $P \leq \psi$ and $(\overline{v} + \underline{v})x - P \geq 0$. Profit will either be $\psi n \int_{\frac{\psi}{\overline{v}+\underline{v}}}^{1} f(z)dz - 3w$ or $(\overline{v} + \underline{v})x^* n \int_{x^*}^{1} f(z)dz - 3w$ s.t. $(\overline{v} + \underline{v})x^* \leq \psi$, which is not the case by assumption, so that profit is

$\psi n \int_{\frac{\psi}{\overline{v}+\underline{v}}}^{1} f(z)dz - 3w$, and the price will be $P_C^{*2} = \psi$. No consumer uses the OSS.

There are therefore two cases, either the OSS developed one feature or two.

In case it developed only one feature, then the entrepreneur has to choose between A, B or C. We have $B > A$ if $\psi > \psi_1$ with $\psi_1 = \underline{v}x^* \frac{1-F(x^*)}{1-F\left(\frac{\psi}{\overline{v}}\right)} \leq \underline{v}x^*$. $C > A$ for $w < w_1$ and $C > B$ for $w < w_2$, with $w_1 = \psi n(1 - F(x^*))$ and $w_2 = \underline{v}x^* n(1 - F(x^*)) - \psi n(F(x^*) - F\left(\frac{\psi}{\overline{v}}\right))$. $w_1 > w_2$ iif $\psi > \psi_1$. Therefore, if $\psi > \psi_1$, C is chosen for $w < w_2$ and B is chosen for $w > w_2$, while if $\psi < \psi_1$, C is chosen for $w < w_1$ and A is chosen for $w > w_1$.

In case the OSS developed two features, then A is never chosen, as it leads to 0 profits, and the comparison between B or C depends on whether w is higher than w_3 defined by the comparison of profits between the two options.

B Proof of proposition 2

By assumption, a monopoly proprietary software develops both features and the interface. This means expected fixed costs of development $2Ec + C$ are not too

high. Since there are no marginal costs of production, the optimal price from the point of view of social welfare is 0. The possible losses in welfare compared to optimality in such conditions thus can have three reasons:

- Less people buy the software, either open-source or proprietary,

  either because it is too expensive (Proprietary) or because it is too difficult to learn (OS);

- More people buy the software, either open-source or proprietary, but for some it has less features than before, or they do not have access to an interface.

- The proprietary software is not developed because the wage w to be paid is too high to justify competing with the OSS. The interface is then not developed, and there is a risk the second feature is not developed due to some mis-coordination in the OS organization.

The first reason can be rejected; $\overline{v}x^* \geq \psi$ which means that the arrival of an OSS, even if it has only one feature, will always increase the number of software users.

The second reason can arise for w and ψ high; the proprietary firm develops the interface but not the second feature, so that its value is lower, but more people have access to the proprietary software: $x_L < x^*$. The effect is ambiguous. In the case where the OSS includes both features,

total welfare will be increased if

$$\int_{x_L}^{x_H} \overline{v} z f(z) dz + \int_{x_H}^{1} [(\overline{v} + \underline{v})z - \psi] f(z) dz - 3Ec - C \geq \int_{x^*}^{1} (\overline{v} + \underline{v}) z f(z) dz - 2Ec - C$$

The left hand side shows welfare from the consumption by lower valuation consumers of the proprietary software, plus the welfare from the consumption of the OSS by high valuation consumers, minus the cost of development (feature + interface for the proprietary software, two features for the OSS). The right hand side shows total welfare when there is a proprietary software monopoly.

It can also happen for ψ low and w high; the proprietary firm develops the two features but not the interface but more people have access to the software. This happens only if the OS software includes only one feature. Total welfare is increased compared to the monopoly case if:

$$\int_{\frac{\psi}{\overline{v}}}^{x^*} (\overline{v} z - \psi) f(z) dz + C \geq \psi \int_{x^*}^{1} f(z) dz + Ec$$

on the left hand-side is the welfare of the additional consumers who have access to the software in its OS form, plus the saving in interface development costs, while on the right hand side is the loss due to the fact the proprietary software doesn't have an interface anymore, plus the additional development cost due to duplication of effort (note that in the OS case, even if the software has got only one feature, there still are two developers who work on it, but in an uncoordinated way).

Finally, when the proprietary development is discouraged, the total welfare may also be lower due to the same kind of effect: higher number of users, but they have to incur more set-up costs. If the OS software includes the two features, for example, total welfare is increased only if

$$\int_{\frac{\psi}{\overline{v}+\underline{v}}}^{1} [(\overline{v} + \underline{v})z - \psi] f(z) dz - 2Ec \geq \int_{x^*}^{1} (\overline{v} + \underline{v}) z f(z) dz - 2Ec - C$$

or

$$\int_{\frac{\psi}{\overline{v}+\underline{v}}}^{x^*} [(\overline{v} + \underline{v})z - \psi] f(z) dz + C \geq \int_{x^*}^{1} \psi f(z) dz$$

i.e. savings in development costs and the additional consumer welfare compensates the consumer's loss from not having access to an interface.

C Proof of proposition 3

The proprietary software will have to include both feature if it is to sell at a positive price. When deciding whether to develop the second feature, OS developers compare the value of the OS software in case it has all features, and thus attracts all users, compared to it value if it has only the first feature and thus attracts a much lower proportion of users. They therefore choose to develop the

second feature if:

$$\overline{v} + \underline{v} + \varphi(n) - c \geq \overline{v} + \varphi(n_1)$$

with n_1 the number of users the OS software attracts when it has only one feature.

n_1 depends on the pricing decision of the proprietary software when faced with an OS software that includes only one feature. The entrepreneur cannot commit on this price, so that there are two cases: either that price is high so that the share of users the OSS gets even when it has only one feature is sufficient to discourage them from developing the second feature, or it is low and OSS developers feel motivated to develop the second feature. In that second case, the proprietary software can sell only if the OSS developers are not put into contact by the OSS organization and thus both develop the first feature only.

The price of the proprietary software will be 0 if the OSS includes both features. If it includes only one feature, then the price will be set to maximize

$$U_E = Pn \int_x^1 f(z)dz - 2w$$

s.t.

$$u(x, \text{Prop}) \geq u(x, OS)$$
$$u(x, \text{Prop}) \geq 0$$

In this case, where there are no set up costs, $u(x, OS) \geq 0$ so that it is the first constraint that binds. It can be rewritten as:

$$x(\overline{v} + \underline{v}) - P + \varphi(n - n_1) \geq x\overline{v} + \varphi(n_1)$$

but $n_1 = nx$, so that we have:

$$P \leq x\underline{v} + \varphi(n - nx) - \varphi(nx)$$

and the entrepreneur maximizes:

$$U_E = [x\underline{v} + \varphi(n - nx) - \varphi(nx)]n[1 - F(x)] - 2w$$

so that x^* is such that

$$\frac{P(x)}{P'(x)} = \frac{1 - F(x)}{f(x)}$$

with

$$P(x) = x\underline{v} + \varphi(n - nx) - \varphi(nx)$$

Does that result in a lower or higher share x of OS users than before? When does the new decision result in a price so low that the OS developers decide to develop the second feature every

time they are put into contact?

Note first that the hazard rate function $r(x) = \frac{f(x)}{1-F(x)}$ is monotone increasing with x because f is logconcave. Suppose $\varphi(x) = kx$. There are positive network effects as long as $k \leq \frac{\underline{v}}{2n}$. $\frac{P(x)}{P'(x)} = x + \frac{kn}{\underline{v}-2kn}$ is increasing with x and the optimal x with network effects will be less than without (when $\frac{P(x)}{P'(x)}$ was equal to x), i.e. the proprietary software will have a higher market share than without network effects.

If f is uniform, the optimal x is 0 for $k \in [\frac{\underline{v}}{3n}, \frac{\underline{v}}{2n}]$, i.e. the proprietary firm takes all the market and set the price $P = kn$. The optimal x is $\frac{1}{2} - \frac{1}{2}\frac{kn}{\underline{v}-2kn}$ for $k \leq \frac{\underline{v}}{3n}$, and the proprietary firm sets $P = \frac{1}{2}(\underline{v} - kn)$.

For $k \in [\frac{\underline{v}}{3n}, \frac{\underline{v}}{2n}]$, an OSS developers with development cost c will choose to develop the second feature if

$$\overline{v} + \underline{v} + kn - c \geq \overline{v}$$

or

$$\underline{v} - c \geq -kn$$

Therefore, if $\underline{v} - \overline{c} \geq -kn$, there is a probability α the OSS includes both features and then the proprietary software has a 0 market share. If not, then the probability the OSS includes both features is the same as without network effects, or $\alpha(1-\lambda)$.

For $k \leq \frac{\underline{v}}{3n}$, an OSS developers with development cost c will choose to develop the second feature if

$$\overline{v} + \underline{v} + kn - c \geq \overline{v} + kn(\frac{1}{2} - \frac{1}{2}\frac{kn}{\underline{v} - 2kn})$$

or

$$\underline{v} - c \geq \frac{kn}{2}\frac{kn - \underline{v}}{\underline{v} - 2kn}$$

Since $k \leq \frac{\underline{v}}{2n}$, $\frac{kn-\underline{v}}{\underline{v}-2kn} < 0$ and the second feature may be developed with probability one but $\frac{kn}{2}\frac{kn-\underline{v}}{\underline{v}-2kn} > -kn$ as $k \leq \frac{\underline{v}}{3n}$ so that this happens for a lesser range of c than when $k \in [\frac{\underline{v}}{3n}, \frac{\underline{v}}{2n}]$.

D Proof of proposition 4

If the proprietary software develops the second feature (the one most valued by the new users), then it has two choices, either set the price such that it is bought only by new users, or lower it such that it is also bought by old users. If it develops the first feature, then it competes frontally with the OSS and will be able to charge a price commensurate with its global market share. For simplicity, assume consumers expect the proprietary software to gain all the market and therefore, a proprietary software gains the whole market when competing with an equivalent OSS. Then, the proprietary software makes a profit of k.[11]

The proprietary software will not want to develop the second feature and sell to all users because that strategy is dominated by developing the first feature and sell to all. Indeed, the price it would be able to charge old users will always be lower than if it develops the first feature, not only because

that feature is not the one they value the most, but also because it will not be able to gain more users than it does when it develops only the first feature (based on the simplifying assumption it can get all the market when competing with the OSS on the same feature), and therefore network effects are no higher.

Therefore, either the proprietary software caters for new users only, and the open-source software serves all the old users and a part of the new (those of low type), or the proprietary software gets the whole market. We will check later that the OSS indeed always develops the first feature.

Profit in the first case is obtained by maximizing:

$$U_E = P(1-s)\int_x^1 f(z)dz$$

s.t.

$$u_P(x,N,2) \geq u_{OS}(x,N,1)$$
$$u_P(x,O,1) \leq u_{OS}(x,O,1)$$

or

$$xv_{2N} + k(1-s)(1-x) - P \geq xv_{1N} + ks + k(1-s)x$$
$$xv_{2O} + k(1-s)(1-x) - P \leq xv_{1O} + ks + k(1-s)x$$

The first constraint will be saturated, which means the second will hold too, and therefore, the entrepreneur maximizes:

$$U_E = [x(v_{2N} - v_{1N}) + k(1-s)(1-2x) - ks](1-s)(1-F(x))$$

If $v_{2N} - v_{1N} - 2k(1-s) \leq 0$ then the derivative of the profit function with respect to x is always negative so that $x^* = 0$. But in that case, $P = k(1-2s)$, and even if this is more than zero, profit is still $k(1-2s)(1-s) < k$ and the entrepreneur thus prefers developing the first feature.

As f is uniform, then $x^* = \frac{1-A}{2}$ with $A = \frac{k(1-2s)}{v_{2N}-v_{1N}-2k(1-s)}$ and for $v_{2N} - v_{1N} \leq k$, $x^* = 1$ so that the entrepreneur doesn't sell.

Therefore, the second feature will not be developed as long as $v_{2N} - v_{1N} \leq \max[k, 2k(1-s)]$.

If $v_{2N} - v_{1N} \geq \max[k, 2k(1-s)]$, then profit when the second feature is developed is $U_E = \frac{(1-s)(v_{2N}-v_{1N}-k)^2}{4(v_{2N}-v_{1N}-2k(1-s))}$ to be compared with profits of k if the first feature is developed. The first feature is therefore developed for $k \in \left[\frac{v_{2N}-v_{1N}}{1-s}\frac{3-s-2\sqrt{s(3-2s)}}{9}, \frac{v_{2N}-v_{1N}}{1-s}\frac{3-s+2\sqrt{s(3-2s)}}{9}\right]$.

But $\frac{v_{2N}-v_{1N}}{\max[1,2(1-s)]} \in \left[\frac{v_{2N}-v_{1N}}{1-s}\frac{3-s-2\sqrt{s(3-2s)}}{9}, \frac{v_{2N}-v_{1N}}{1-s}\frac{3-s+2\sqrt{s(3-2s)}}{9}\right]$ so that the second feature is developed only for $k \leq \frac{v_{2N}-v_{1N}}{1-s}\frac{3-s-2\sqrt{s(3-2s)}}{9}$.

Verify also that the open-source organization will always prefer to develop the first feature rather than the second, because this always will be in the best interest of its constituent. Indeed, developing the second feature only means the old users get less value from the software: its value is lower than the first feature's value, and the OSS in that case will not get as many users than if the first feature had been developed.

E A brief history of the development of TEX and its competitors

For dates, the reading of that chronology should be accompanied by the reading of the graphical chronology of the development of TEX p.37[12]. That chronology divides the development of TEX according to a typology (core, macro, organization) that is explained later. It also presents the chronology of the development of alternatives to the open-source TEX system. That is divided in four parts according to the typology exposed in the table 1 p.10.

The development of the TEX system can be divided in three areas: development of the core, development of macros on top of the core, and the development of the infrastructure allowing easier use and development of the TEX system. TEX itself went through three versions, all developed by D.E. Knuth. TeX2C, renamed Web2C, allowed to translate the TEX source code into C, the dominant programming language. This provided the basis for most subsequent TEX implementations and distributions (more on that later). Later on, as the development of TEX was frozen, various projects came about to rewrite and expand the core: eTEX, NTS, pdfTEX, Omega, with pdfTEX being currently the most used as it corresponds to the general need to generate pdf files with TEX.

Macro development on top of TEX came about very rapidly after the release of the first versions of TEX, as each author was supposed to develop its own set of macros. LATEX by Lamport came to be the dominant macro package as it fit the needs of most users. Its development was taken over by the LATEX3 team which developed LATEX 2_ε, while Hans Hagen developed his own set of macros, ConTEXt.

The TUG was first set up to diffuse information about TEX and encourage meeting of developers and the use of the program. It is only in the 90s though that it set up a common repository for TEX packages (CTAN) and a common directory structure for TEX distributions (TDS). TEXLive, the official open-source LATEX distribution was based on Web2C and the teTEX distribution, which was translated for Windows systems into fpTEX. Independently of that community effort, the MikTEX Windows distribution was very popular. Interfaces were developed, the most popular being WinShell. Further in the effort to standardize the OS LATEX offering, the LATEX Project Public License was adopted for most packages in a LATEX distribution. In order to facilitate the inclusion of TEX systems into the Debian Linux distribution, it was slightly changed to conform to the Debian OS guidelines.

Competition had various faces: prior to TEX there existed proprietary typesetting systems (Script, Scribe, roff) of which only roff had a lasting legacy. The main competitors for TEX's clientele came in the middle 80s, with MSWord taking up the less sophisticated users while publishers

	Core	*Macro*	*Organization*	*Competing Proprietary*	*Competing OS*	*Inspired Proprietary*	*Inspired OS*
1970				Script by IBM			
1971				Roff as part of Unix			
1972							
1973							
1974							
1975							
1976							
1977							
1978	TeX 1.0 in Sail			Scribe			
1979							
1980			TUG				
1981							
1982	TeX 2.0 in Pascal						
1983		LaTeX, AMS-TeX					
1984		BibTeX		MS Word			
1985		LaTeX 2.09		Pagemaker, postscript by Adobe		PCTeX, MicroTeX	
1986				Framemaker 1.0, Ventura, 3B2	SGML		
1987	TeX2C, TeX in C			QuarkXPress			
1988				MS Word for Windows			
1989	TeX 3.0, NFS	LaTeX3 team				TeXtures	
1990	Web2C	AMS-LaTeX	CTAN				
1991					Groff, roff for GNU	Y&Y	
1992	eTeX initiated					Scientific Word	
1993	NTS initiated		4AllTeX	pdf by Adobe			
1994	Omega initiated	LaTeX2e, ConTeXt	TDS, teTeX, Bakoma				
1995					HTML 2.0	WinEdt	Lyrix (then Lyx)
1996	eTeX released, pdfTeX		MikTeX, TeXLive				
1997							
1998	NTS released		WinShell, fpTeX		XML v1.0, Corel WordPerfect for Linux		
1999			LPPL				LyX 1.0
2000				InDesign by Adobe	XHTML 1.0		
2001							
2002			Debian guidelines		Abiword 1.0, OpenOffice		TeXMacs 1.0

Figure 2: Chronology of the development of TEX

adopted QuarkXpress, Framemakers and other proprietary systems. Those systems were relatively easy to learn, were self contained and were able to establish themselves as standards in the industry. QuarkXpress particularly became the equivalent of the QWERTY keyboard in the industry: not necessarily the most efficient or powerful, but any typesetter knew how to use it. Adobe bought Pagemaker in 1994 and Framemaker soon after and gradually imposed its standards (postscript, then pdf) to the industry. It launched its own desktop publishing application, Indesign, in 2000.

Another type of "competition" for OS TEX systems came from proprietary typesetting systems based on TEX. PCTEX, Y&Y, TEXtures, Scientific Word, ... thus based their product on proprietary TEX implementations, and were able to sell because of their better fonts (Y&Y), interface (WYSIWYG interface of Scientific Word) or adaptation to a platform that was neglected by the OS community (Macintosh by TEXtures, personal computers by PCTEX in its time). WinEdt only provided an interface to standard OS TEX distributions and was sold at a much lower price than those proprietary implementations of TEX.

Open-source competition was mainly in promoting alternative typesetting languages than LATEX, enabling typesetters to have a wider latitude in formatting their text. This led to an effort in the LATEX community to make LATEX SGML, then XML compliant. Direct OS competitors to mainstream TEX were inspired by it, such as LYX or TEXMacs, which were variations on the concept of WYSIWYG interfaces to TEX and were generally less flexible than a full TEX distribution but easier to use and learn.