2006RP-03

The Economics of Free and Open Source Software: *Contributions to a Government Policy on Open Source Software*

Marcel Boyer, Jacques Robert

Rapport de projet
Project report

Part of a RESOLL study conducted and translated for the Treasury Board of Canada Secretariat

Montreal
February 2006

CIRANO
Centre interuniversitaire de recherche en analyse des organisations

ISSN 1499-8610 (Version imprimée) / ISSN 1499-8629 (Version en ligne)

The Economics of Free and Open Source Software: *Contributions to a Government Policy on Open Source Software*

Marcel Boyer[1], *Jacques Robert*[2]

Résumé / *Abstract*

This document seeks to lay the groundwork for a government policy on free and open source software. We briefly characterize the extent of the open source software phenomenon. We analyse its pros and cons for the government, in its role as both an engine of economic development and a large user of information and communications technologies. We conclude with a series of recommendations for the government, as both "economic and industrial policy maker" and "large user."

Keywords: free software, intellectual property rights, free source code, open source code, free operating system, GPL licence, BSD licence, innovation, forking

[1] Bell Canada Professor of Industrial Economics, Université de Montréal, C.D. Howe Scholar in Economic Policy, Fellow, CIRANO and CIREQ. Address: CIRANO, 2020 rue University, 25e étage, Montréal (Québec), H3A 2A5; e-mail: marcel.boyer@cirano.qc.ca
[2] Fellow CIRANO and Department Information Technologies, HEC Montréal, e-mail: jacques.robert@hec.ca

This research paper does not purport to be an exhaustive review of the debates, analyses, or issues associated with the development and use of free and open source software (FOSS)—software whose source code is freely available and that can be used without licence fees. Many relevant issues exceed the scope of our study. Our goal is to provide a neutral and balanced evaluation of the prospects offered by the use of free and open source software within public administrations using economic analysis—or, more precisely, the economics of organizations (motivation, incentives, and coordination) and industrial economics. In this manner we hope to bring a constructive and illuminating contribution.

The current document is divided into five sections. In the introduction, we present the broad goals of the study and a brief history of the development of free and open source software. In Section 2, we describe the criteria that define free and open source software and provide a snapshot of the various licence types under which it is released. Section 3 is devoted to the advantages and disadvantages of *using* (demand) free and open source software. In Section 4 we examine the issue of *innovation* (supply) and demonstrate how both a regime based on the protection of proprietary software and one based on recognizing free software can promote innovation and the creation of value, each in its own way. Section 5 is devoted to the economic principles of particular relevance to the choice between proprietary and free software confronting a government preoccupied by both its interests as a user and its duty as an engine of the economic development and expansion of the software and IT services industry. We conclude with a number of recommendations for government policy with respect to proprietary and free software.

1. Introduction

The use of open source software is becoming increasingly widespread. Major initiatives, such as Linux, Apache,[3] and OpenOffice (to name but a few) have allowed the creation of software whose code is accessible, free, and open to analyses and contributions from peers. Until recently, open source software was only of interest to a small community of programmers and developers. Today these computer programs have become a serious alternative to proprietary software.

The very existence and proliferation of this software casts doubts on the principle that the creation of value is maximized by protecting proprietary systems through copyright—a principle based on the assumption that copyright protection is necessarily more effective. This paper will seek to better understand the phenomenon of free and open source software from the perspective of an economic analysis and to derive recommendations with regard to government policy on free software.

1.1 Overall objectives

As users of software, it is increasingly incumbent on governments to account for developments in free software, to the extent that it represents real alternatives to proprietary systems. Among the documented advantages of free software, we note that it provides less expensive solutions that are more robust and secure in some cases, and grants its users greater contractual independence from software publishers and more protection from opportunistic behaviour when contracts are renewed or the software updated. Furthermore, government users can more easily opt to call on local resources without forfeiting the cost advantages (economies of scale, size, forking, and network) of which some proprietary software publishers can rightly boast.

[3] The Linux operating system (named as an amalgamation of its designer's name, the Finn Linus Torvalds, and the operating Unix on which it is based), and the Apache Web server (ostensibly derived from "a patchy server") are two examples of free software.

Free software also raises political economy issues. The use of free software by governments has an effect on, and contributes to, the structural transformation of the software and computer services industries. It will be no surprise to learn that the development of free and open source software was largely a reaction to the market power of large proprietary software publishers, and it is specifically designed to modify competitive relationships within the software industry and to bolster local computer services.

It appears obvious that the production of creative intellectual work requires adequate copyright protection. This is an important condition to sufficiently motivate authors, actors, and other social and economic agents to invest their time and resources in the production of such works. This mainstream approach is based on the general argument that private intellectual property, protected by an effective regime for recognizing copyright (like the clear and enforceable physical property rights regime) allows wealth creation to be maximized by giving creators the means to capture a significant share of the value created.

But this is not the only approach open to us. Under certain circumstances, we recognize the importance of free dissemination and movement of ideas and intellectual contributions. To a large extent, modern governments promote this free movement and recycling of ideas by directly using their resources and fiscal powers to encourage the development of science and research. Collectively, we all benefit from the spin-offs (spillover effects) of this growth in knowledge, referred to as public domain. Thus, two approaches or visions clash: one in which intellectual contributions are protected by copyright, and another in which these contributions are “recognized,” but with a concomitant expectation that the creations will be surrendered to the public domain.

Free software must be assessed on its merits, just like any proposed alternative solution. Ultimately, the choice of technological solutions must be founded on case-by-case evaluations. However, we believe that there are particularities specific to free software that requires a more thorough assessment and a better understanding.

The breadth of the free software phenomenon raises an inescapable paradox: How can we explain that developers and private firms invest resources, often in large volumes, to develop software for free distribution? Among the explanations proffered in the academic literature, we notice in particular the desire of developers to establish their reputation and promote their skills, as well as the difficulty of developing complex software in isolation. In fact, we realize that the software industry has unique features, which may make free software economically viable and profitable for both developers and users. This industry is characterized by significant network effects, economies of scale and size, pronounced forking, an insatiable appetite for innovation, and substantial switching costs.

In summary, software is a very complex product that is constantly mutating and needs to meet high compatibility standards. Developing and imposing significant proprietary standards is beyond the scope of any one firm, except the very largest…or even THE very largest. This is why companies like HP, IBM, and Sun invest large amounts in the development of major open source initiatives.

1.2 A brief history of the free and open source software phenomenon

Software can be distributed as either source code or binary code. Source code can be understood and modified by an analyst, while binary code is a series of 0-s and 1-s that are virtually inscrutable to humans. Thus, it is possible to protect software by concealing its source code. Most commercial software is distributed solely as binary code—when the source code is made available, it is accompanied by a licence greatly restricting its use.

This was not always the case. Initially, operating systems were developed within universities, such as Berkeley and MIT, or large private research centres (Bell Labs and Xerox' Palo Alto Research Center). Programmers in these research centres voluntarily shared the source code they were working on.

Simultaneously, we witnessed the emergence of the personal computer industry. Designing and commercializing software became a profitable enterprise, and large companies arose that made fortunes selling proprietary software. When AT&T began to enforce its intellectual property rights over UNIX in the early 1980s, a movement of resistance began to coalesce.

The Free Software Foundation (FSF)[4] was founded by Richard Stallman of MIT's Artificial Intelligence Laboratory. The purpose of this Foundation is to promote the development and dissemination of free software. As it states on its Web site, the FSF "*is dedicated to promoting computer users' rights to use, study, copy, modify, and redistribute computer programs*." The foundation offers the GPL (General Public License) and an operating system called GNU.[5] The purpose of the GPL is to require that (a) the software source code be distributed free of charge and (b) any user who uses and modifies the source code must, in turn, make the redistribution and its source code available free of charge. Despite the fact that this vision is difficult to reconcile with the development, dissemination, and commercialization of software, the idea of developing and enhancing software on the basis of cooperation and communitarianism has proven increasingly popular.

In tandem with the development of the GPL and the GNU operating system, we have observed the appearance of various forms of free software. "Shareware," for example, is software that is freely distributed (generally for a test period).[6] Unlike in the case of free software, the source code of shareware is not usually distributed. The BSD (Berkeley Software Distribution) licence offers an alternative approach to the GPL. The BSD allows anyone to copy and modify the source code, but also allows those who do so to appropriate their modifications and improvements in order to commercialize them, thus refraining from making them public.

The beginning of the 1990s was marked by the appearance of the World Wide Web and a surge in activity around free software. In 1991, the Finn Linus Torvalds took upon himself to create an operating system to replace UNIX. After several months of solitary effort, he made

[4] Free Software Foundation: http://www.fsf.org.
[5] GNU is a recursive acronym for "GNU's Not UNIX."
[6] Despite some similarities, it is important not to confuse free software with freeware, shareware, or public domain software. In our context, free means "unrestricted" and not "costless," even though currently most free software is distributed at no, or at a paltry, cost. Free software is protected by copyright. Free software is also called open source software, to emphasize that the source code is publicly available.

his source code available and extended an invitation to anyone wishing to suggest additions and improvements. To his great surprise, there were a large number of takers. Recall that this was at the dawn of the era of the Web. The success of Linux has been phenomenal. It has become one of the main operating systems for servers and the only one able to compete with MS Windows on personal computers. Today, the development of Linux is supported by OSDL (Open Source Development Labs), a non-profit consortium, among others. The list of OSDL members includes large high-tech firms, such as IBM, Cisco, HP, Sun Microsystems, Novell, Intel, Red Hat, and Google.

As of 2000, the massive entry of large information technology corporations into supporting free software has been its most striking aspect. Today there are few computer applications with no corresponding free software initiative. While Linux provides a credible alternative to MS Windows, OpenOffice.org seeks to replace MS Office. Mozilla's Web browser, FireFox, rapidly appropriated a large segment of Internet Explorer's market. And the list goes on: MySQL (database management), Thunderbird (e-mail), Spybot (privacy software), Apache (Web server), Compiere (SME ERP system), etc.

Today, the world of free software is more dynamic than ever. The SourceForge.net Website, the largest repository of open source software on the Internet, has a catalogue of over 100,000 projects and more than one million registered members. Of course, not all of these projects are on the scale of Linux, but many are sufficiently mature to have generated high-quality software. The emergence of free software is no longer merely a marginal and paradoxical epiphenomenon, but rather a well-established reality—ubiquitous and flourishing.

2. FOSS, definition, criteria, and licence types

On its Web site, the Open Source Initiative (OSI) presents a comprehensive definition of free and open source software (FOSS). It is not enough for the source code to be open, other criteria must be met as well. The Open Source Initiative presents a list of ten of them.

TABLE 1: The 10 Criteria of the Open Source Initiative

(Source: http://www.opensource.org/docs/definition.php)

1. *Free Redistribution – (...) The license shall not require a royalty or other fee for such sale.*
2. *Source Code- The program must include source code, and must allow distribution in source code as well as compiled form. (...)*
3. *Derived Works- The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software. (...)*
4. *Integrity of The Author's Source Code – (...) The license may require derived works to carry a different name or version number from the original software.*
5. *No Discrimination Against Persons or Groups - The license must not discriminate against any person or group of persons. (...)*
6. *No Discrimination Against Fields of Endeavor - The license must not restrict anyone from making use of the program in a specific field of endeavor. (...)*
7. *Distribution of License - The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties. (...)*
8. *License Must Not Be Specific to a Product - The rights attached to the program must not depend on the program's being part of a particular software distribution. (...)*
9. *License Must Not Restrict Other Software - The license must not place restrictions on other software that is distributed along with the licensed software. (...)*
10. *License Must Be Technology-Neutral - No provision of the license may be predicated on any individual technology or style of interface. (...)"*

There are a number of licences that satisfy the 10 criteria presented in Table 1. Each one has unique features. Our goal here is not to present a comprehensive description of each of these licences, but rather to provide a succinct summary of their principal differences. The table below, reproduced from the MITRE study (2003), compares the rights and obligations specified by the different free software licences.

Table 1. A Comparison of FOSS and Related Licenses

Property / License:	GPL	LGPL	BSD & MIT	Apache	Public Domain	Microsoft MIT[4] EULA
a. Can be stored on disk with other license types	✓	✓	✓	✓	✓	(bans FOSS)[5]
b. Can be executed in parallel with other license types	✓	✓	✓	✓	✓	(bans FOSS)[5]
c. Can be executed on top of other license types	✓	✓	✓	✓	✓	(bans FOSS)[5]
d. Can be executed underneath other license types	✓[1]	✓	✓	✓	✓	(bans FOSS)[5]
e. Source can be integrated with other license types		✓	✓	✓	✓	(bans FOSS)[5]
f. User decides if and when to publish derived code	✓[2]	✓	✓	✓	✓	✓
g. Software can be sold for a profit	✓	✓	✓	✓	✓	✓
h. Binary code can be replicated by users as desired	✓	✓	✓	✓	✓	
i. Binary code can be redistributed as desired	✓[3]	✓	✓	✓	✓	
j. Binary code can be used as desired by users	✓	✓	✓	✓	✓	
k. New users always receive source code of derived works	✓	✓[6]				
l. New users receive full source modification rights for derived works	✓	✓[6]				
m. New users receive full redistribution rights for derived works	✓	✓[6]				
n. Binary code can be released without source code			✓	✓	✓	✓
o. Derived code can have a different type of license		[7]			✓	
p. Original source can be incorporated into closed source products					✓	

[1] *Provided that both programs are fully and independently usable in other unrelated contexts.*
[2] *Provided that the binary code has not been previously released to the public.*
[3] *Provided that source code is always redistributed along with the binary code.*
[4] *The proprietary Microsoft MIT EULA is not related to the similarly named MIT (X/MIT) license.*
[5] *Specifically bans use of: GPL, LGPL, Artistic, Perl, Mozilla, Netscape, Sun Community, and Sun Industry Standards.*
[6] *The rights granted by LGPL do not necessarily extend to the applications linked into an LGPL library.*
[7] *The LGPL does permit re-licensing under GPL as a special case, but not re-licensing under any other license type.*

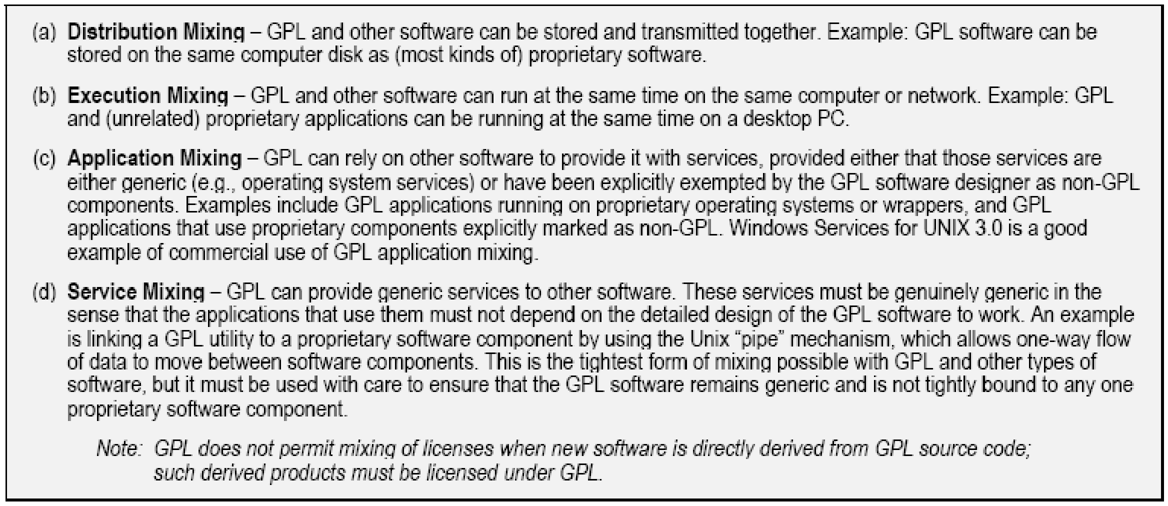
License Acronyms:

GPL –	*GNU General Public License*	*(Microsoft) MIT –*	*Mobile Internet Toolkit*
LGPL –	*GNU Lesser General Public License*	*(X/MIT) MIT –*	*Massachusetts Institute of Technology*
BSD –	*Berkeley Software Distribution*	*EULA –*	*End-User License Agreement*
MPL –	*Mozilla Public License*	*FOSS –*	*Free and Open-Source Software*

The different licences in the free and open software world are distinguished in terms of the freedom they grant to reuse the source code by modifying it and incorporating it into new products. The GPL explicitly prohibits redistributing the software or modified versions thereof without the source code. The GPL is called a viral license, since it forces all modifications to the source code to be released under it as well. Licences of the BSD or Public Domain type are less restrictive in this respect.

In practice, it is indeed an important issue whether free software can be associated with, or linked to, proprietary systems. In general, the answer is yes, a private firm may reuse source code emanating from the free community in the creation of a new product. Some limitations exist, however, depending on the licence. The following figure, from MITRE (2003), reveals the options a firm has when associating GPL code with proprietary source code.

Figure 1. Strategies for Mixing GPL and Proprietary Software



(a) **Distribution Mixing** – GPL and other software can be stored and transmitted together. Example: GPL software can be stored on the same computer disk as (most kinds of) proprietary software.

(b) **Execution Mixing** – GPL and other software can run at the same time on the same computer or network. Example: GPL and (unrelated) proprietary applications can be running at the same time on a desktop PC.

(c) **Application Mixing** – GPL can rely on other software to provide it with services, provided either that those services are either generic (e.g., operating system services) or have been explicitly exempted by the GPL software designer as non-GPL components. Examples include GPL applications running on proprietary operating systems or wrappers, and GPL applications that use proprietary components explicitly marked as non-GPL. Windows Services for UNIX 3.0 is a good example of commercial use of GPL application mixing.

(d) **Service Mixing** – GPL can provide generic services to other software. These services must be genuinely generic in the sense that the applications that use them must not depend on the detailed design of the GPL software to work. An example is linking a GPL utility to a proprietary software component by using the Unix "pipe" mechanism, which allows one-way flow of data to move between software components. This is the tightest form of mixing possible with GPL and other types of software, but it must be used with care to ensure that the GPL software remains generic and is not tightly bound to any one proprietary software component.

Note: GPL does not permit mixing of licenses when new software is directly derived from GPL source code; such derived products must be licensed under GPL.

From an economic perspective, this question is important. Licences attached to free software must not stifle all forms of private initiative in terms of value creation. In practice, creating, improving, and supporting a computer application based on free software must be a profitable proposition. Many firms have, in fact, succeeded in rising to the challenge and developed profitable business models while dealing with free software. Red Hat sells support and documentation for Linux; MySQL AB is a private firm supplying training, support, and consulting services around the free database system MySQL; and many companies, notably Google, use free software to distribute advanced services on the Web.

3. Advantages and disadvantages of FOSS

One of the benefits of free software is that it is usually founded on open standards. An open standard is a protocol that defines the specifications for interfacing between software, routines, and subroutines (bits and pieces of software) to form a consistent and operational whole. The presence of open standards makes it possible for anyone to develop code that can be integrated into existing code. The use of open standards is a key element of any long-term strategy. In a report to the Danish government (October 2002), the Danish Board of Technology states:

> "*A strategy for e-government should not be based on a closed, proprietary standard in a key technology. The first reason for this is that it is unacceptable as a matter of principle for enterprises and citizens not to be able to choose between different suppliers of the software that is necessary to use the services of public authorities that are offered in the form of e-government. The second is that it is vital to the socio-economic cost-effectiveness of far-reaching e-government that a competitive situation can be established that ensures the presence of competing products. A condition that must be met for this to be achieved is that open standards are used* "

The development and maintenance of proprietary software are (exclusively) the province of the owner of the code or a licensed agent. In the case of free software, this development occurs spontaneously in accordance with the needs of the user community and by relying on open standards. This decentralized approach to creating free software is based on and contributes to the following processes and characteristics:

(i) free software is modular;
(ii) its development is based on open standards;
(iii) the predominantly modular design of free software allows partial installations according to need;
(iv) the use of open standards allows modules to be implemented with a broad range of complementary software using the same standards;
(v) finally, free software is constantly evolving.

a) Costs

The primary benefit of free and open source software is its low cost. It can often be obtained for nothing, since it may be downloaded free of charge from the Internet. Lerner and Tirole

(2000) underscore that the use of free and open source software can also shield firms from monopolistic behaviour and escalating update costs. Dale et al. (2004) mention that open source software tends to contain fewer lines of code and to be modular, thus allowing it to work on less powerful computers and contributing to lower systems costs. Kenwood (2001) states that free software provides firms using it with an improved price-performance ratio. He believes that this software allows the costs of routine and administrative costs to be cut. It also allows support, installation, integration, and generally the implementation of systems to be less expensive, while simplifying customization and updating. The source code of this type of software is generally less bloated and so requires fewer resources to function effectively.

A study conducted by MITRE Corporation (MITRE, 2003)[7] for the U.S. Department of Defense sought to assess the impact of abolishing free and open source software in the Department. The analysts concluded that this type of software plays a vital role in the military on several levels. While the software market offers commercial software that features essentially the same characteristics as open source software, abolishing the latter would result in a sudden jump in short-term costs, with free network and Web application software being replaced with equivalent proprietary software. Research would also be severely affected by the elimination of open source software. Owing to budget constraints, researchers in the Department use free software, allowing them to focus on the core of their research more rapidly.

b) Forking

One of the greatest, but least known, benefits is the forking effect. This effect measures the value to a firm of guiding the development of free and open source software by adding a branch or a new sub-routine (path) that completes the current software and then inviting developers to improve this new path. Once the new path has integrated improvements from external developers, the firm can benefit from these updates by incorporating the developments and enhancements into its in-house use of the software as desired. This forking

[7] See also the critique of this report by the ISC (Initiative for Software Choice): http://softwarechoice.org/download_files/MITRE.Final.Web.pdf.

effect is reminiscent of a real option created by the company and potentially proving very profitable in the future. In the previously mentioned IDC status report, Techworld makes the following points:

> « *Another surprise was that many companies said the ability to customise open-source software was important. IDC didn't suggest this as one of the standard multiple-choice answers. Instead, many companies added it in the "comments" section of the survey. Vendors of pre-packaged, proprietary software routinely downplay the customisability of open-source, arguing customers are not interested in extending software themselves. "These companies don't want to start building an application from scratch, but they can build their own additions to an already-complete application," [analyst Bo Lykkegaard] said. "Because they are part of an open-source community, they can feed this back into the software and it can become a part of the next release, meaning people are helping you to maintain your customisations." He said many companies turn to customisations when they can't find commercial software that meets their needs.* »

This forking effect complements the economies of scale, scope, and network that have traditionally been associated with software development. These economies are found throughout the software industry, whether proprietary or free.

c) Flexibility and modularity

One of the principal benefits of free software is the combination of flexibility and modularity. Flexibility assumes different forms: flexibility in adapting, flexibility of shared systems, interoperability between systems, and the choice of an integration services provider. As to modularity, it allows sophisticated users to add modules designed to meet specific needs to those already available and, in turn, to release the new modules to the community in order to facilitate further development (forking), from which they may benefit at an opportune moment.

Access to the source code makes modification, improvement, and customization possible. An increasing number of firms assert that the choice of free and open source software is motivated less by its lower price but more by its vector of characteristics providing better performance and higher quality. A recent IDC survey[8] on the penetration of free software into European firms reveals that those who consider the quality of their software to be a determinant factor in their competitiveness—such as firms active in telecommunications, financial services, and business services—are more likely than others to opt for free software for reasons of quality and flexibility rather than simply because of cost. In reality, firms are thus building a portfolio of real options[9] that may prove very valuable, especially in an environment characterized by great uncertainty, volatility, and ignorance regarding the future, irreversibility of significant costs (it is very expensive to backtrack), and considerable obstacles (costs) to adapting or modifying software. Indeed, the value of real options, as represented and created by the forking, flexibility, and modularity associated with free software, is directly proportional to the level of uncertainty and irreversibility.

Since users are not homogenous, this flexibility allows software to be tailored to the specific needs of the client (Lerner and Tirole, 2000; GBdirect[10]). It is possible to eliminate functionalities that are superfluous for the user while adding others (Kenwood, 2001; Dale et alii, 2004; Kuan, 2000).

Software publishers seek to link specific applications with their whole menu of products. To the extent that free software is not the creation of a single manufacturer, it becomes possible to integrate it into non-proprietary technological platforms. After adopting this technology, users of free software are no long beholden to developers or bound by their price structures (Lerner and Tirole, 2000). Consequently, users can call on several integrators to compete for their business. For these users, free software allows "lock-in" to be avoided. A sophisticated user, such as a government body, can then design a multi-sourcing strategy, relying on both

[8] IDC, Western European End-User Survey: 2005 Spending Priorities, Outsourcing, Open Source, and Impact of Compliance, quoted by Techworld (www.techworld.com). IDC, a subsidiary of IDG (International Data Group), is the world's largest company specialized in consulting and analysis related to IT markets.
[9] Cf. Boyer, Christofferson, Lasserre and Pavlov (2003) for an introduction to real options.
[10] Gbdirect (www.ebusiness.gbdirect.co.uk) is an e-commerce and IT consulting firm. See, in particular http://open-source.gbdirect.co.uk/migration/benefit.html.

in-house and external sources, to increase flexibility in procurement, reduce costs, and improve the quality of its software and systems.

A government seeking to foster industrial development can capitalize on these characteristics to promote the emergence of greater competitiveness on the market for IT support services (integration and others), without forfeiting the economies of scale, scope, forking, and network that are specific to this industry. Under certain conditions, free and open source software may increase these economies. The maturity and reliability of free software can only be enhanced by an expansion of the user community. The free software community is particularly vulnerable to the process of "natural selection," ensuring ongoing improvements in the software.

d) Risk and risk management

In the world of software, risk has many faces. Various sources of risk interact, exacerbating or neutralizing each other. It is important to have a global, integrated perspective on these sources of risk and their interactions. Even the notion of risk itself is not always well understood. A risk exists when the performance of the software depends on exogenous factors that are more or less random and beyond the control of the decision maker responsible for choosing the software or the manager responsible for its performance. These external and random factors may cause the software to perform better or worse than expectations or averages would justify. This volatility should be treated as a characteristic of the distribution of performance around the mean expectation of the decision maker or user, but it is often perceived as a measure of the probability that the performance will *fall short of* a critical threshold located below the anticipated level. Drawing on the terminology of finance, we may speak of "performance-at-risk," i.e. a performance level more than one or two standard errors below the expected performance. In other words, the value assigned to the critical performance threshold is such that the probability of the actual level falling below it would be 0.10 or 0.05. Obviously, characterizing this performance-at-risk or threshold value quantitatively is no simple matter. However, it does have the advantage of corresponding to a well-known concept of risk.

What are these exogenous and stochastic factors that may impact on the expected performance of software? Following is a (partial) list: in-house capabilities and the organization's competencies in manipulating and integrating software; the degree of sophistication and competitiveness of the service provider market; warranties or penalties on the supplier, or costs it must bear in the event of poor performance; and the stage in the lifecycle of the product's or software's development. These factors can be assessed before making any important decisions. They apply to both proprietary and free software.

The measures available to the organization for controlling and reducing these risks assume various forms. We have both *ex ante* measures (self-protection measures), designed to increase the likelihood that the software will deliver the promised level of performance, and *ex post* measures (self-insurance measures), designed to limit the fallout should the software underperform catastrophically. Naturally, these measures will depend on whether the software is proprietary or free.

We frequently hear that managers and decision makers responsible for the implementation and performance of free software tend to feel trapped by internal incentive mechanisms (compensation, bonuses, and promotions) that bias decisions in favour of minimizing risk taking and promoting the status quo—to the detriment of free software. Indeed, if proprietary software performs poorly or is inappropriate for the tasks for which it was acquired, the manager having made the original decision can take the matter up with the owner-supplier of the program. In the case of free software, shifting blame like this is generally not an option. For the best software, proprietary or free, to actually be chosen, this phenomenon of perverse incentives must be overcome.

e) Protection and security

The MITRE (2002) study for the U.S. Department of Defense points out that important elements of the Department's security depend on the use of free software. For historical reasons, the first software created in the free and open source community was designed to carry the Internet—including security software. Free and open source software was created to run use in a multi-user environment and its security features were designed with this in mind (YLEM).

The fact that the code is open creates benefits and disadvantages. On the one hand, this makes it possible for hackers to easily examine the source code and find security holes, but on the other hand it also makes it easier for programmers to quickly identify and solve problems. When a bug is discovered, it is rapidly corrected and a patch made available almost immediately. Since upgrades are free, users can avail themselves of the repaired versions without waiting for an official release. Improvements and innovations are always accumulated in the most recent version of the software—available to all.

When needed, a user can always apply static analysis tools to detect the presence of hostile code (Kenwood, 2001). Thus, automated tools can be applied to reduce the level of effort devoted to searching for vulnerabilities (Dale et alii, 2004).

As previously mentioned, one of the most surprising results of the MITRE (2002) study is the degree to which the security of the Department of Defense depends on free and open source software. Eliminating this type of software would thus have an immediate, negative, far-reaching, and in some cases very severe impact on the military's ability to monitor its own network and protect it against intruders. Such an abolition would also undercut its ability to modify the infrastructure source code rapidly in response to new types of cyber attacks.

The free and open source software community further contributes to the Department's security in two additional manners. First, it creates software like OpenBSD, which combines a very low failure rate with rapid sealing off of security breaches. Second, the community has long been fascinated with the development of more sophisticated applications for identifying and analysing security holes in networks and on personal computers.

This software type also allows a form of active publication used by researchers to share, not results, but the actual software, which can immediately be put to use in their research efforts.

The elimination of open source software would thus have a short- and long-term impact on the system's capacity and security as researchers would lose access to significant resources.

Most of the Internet's infrastructure having been created on the model of free and open source software, it is usually older and more mature in terms of functionalities, as well as more stable, than the bulk of the equivalent proprietary software currently available.

f) Lifespan, lifecycle, and stability

Not all free software is reliable and effective. Many free software projects disappear on their own, failing to elicit enough interest in the community to sustain their development. Nonetheless, when a sufficiently large community adopts a free software project, its development may be ensured by this community even when the initial creator drops out. Generally, this is not the case with proprietary software, which will sometimes be discontinued so that a new version can be brought to market. In the case of free software, however, the source code remains available, so developers can continue improving it. Its lifespan will often exceed that of proprietary software.

g) Support and documentation

Commercial software is sold with high-quality tutorials and software publishers often provide professional customer support services. In the field of free software, the quality of support is spottier. Free and open source software is typically developed by sophisticated users. Consequently, the interface is not always very user-friendly. Blanas (2003) asserts that there are not enough high-quality reference manuals to support the training of free and open source software users. Users are often on their own in matters of deciphering the code and understanding how it works.

In other cases, however, there exist a plethora of reference works. Support and documentation is an industry in its own right, and can be very profitable. In some cases, the business model of those who give away their software consists of selling services of documentation, training, and support. Ghosh (2002) maintains that the users of free software are generally prepared to accept inadequate documentation and a less sophisticated interface in exchange for lower costs and the possibility of modifying the code to adapt it to specific needs. When additional support becomes necessary, this can be obtained from an external supplier who could be totally unaffiliated with the original developer.

It should also be pointed out that free and open source communities are based on a certain level of reciprocity. Members of the community can solicit help from others, who will come to the rescue. Contributions made to the advancement of a project create a sense of gratitude in those who are committed to it, and they will in turn be prepared to respond to calls for help.

We may conclude our comments on support and documentation with the following remark drawn from the Gbdirect Website:[11]

> « *Detractors of open-source software quite rightly point out that the free licence to use the software includes no support contract. But they neglect to mention the other side of that issue: many proprietary software licences have no support included either. Indeed, the majority of mass-market proprietary software support is aimed at hand-holding for inexperienced users. Just as proprietary vendors will sell support contracts with agreed service levels, suppliers and third parties will provide support for open-source software. An example of this is the Gnat translator for the Ada programming language, as reported in [CONECTA2000] and elsewhere.*
>
> > *"ACT Europe was founded in 1996 to provide support for commercial, industrial and military uses of the GNAT Professional Ada 95 28 development environment in Europe. It was founded jointly by Ada Core Technologies Inc (ACT), and by the European members of the GNAT Ada 95 project. ACT is a privately held corporation and was founded in August 1994 by the principal authors of the GNAT Ada 95 compiler*

[11] http://open-source.gbdirect.co.uk/migration/benefit.html

system. ACT was founded without any initial capital other than some small loans from the principles, and has existed entirely from revenue from its inception. ACT claims to be currently profitable. The people involved in both ACT and ACT Europe (from now on, ACT) have been working with Ada for over twenty years, starting with the development of a working Ada compiler for preliminary Ada in 1979, and the first validated Ada 83 system in 1983. This work was done at New York University by a team dedicated to the technical success of the Ada language, which moved to ACT after its foundation. GNAT is the most widely used Ada 95 development system, available on many platforms, from workstations and PCs to bare boards. Ada solutions using GNAT encompass conventional, real-time, embedded, and distributed systems applications. The GNAT technology has always been based on free software, and more specifically on the GNU toolset. The GNAT compiler is integrated with the GCC (the GNU C compiler) back-end. The GNAT debugger is based on GDB (the GNU debugger) that has been adapted for Ada 95. GLIDE, the GNAT integrated development environment, is based on Emacs, which has been adapted and complemented to create a friendlier and complete environment. The GTKAda GUI technology leverages on the GTK graphical toolkit and provides an advanced GUI builder. These are a few but significant examples of the technological offering of ACT. All the products being distributed by ACT are free software, usually under a GPL or LGPL-like licence. Since its foundation, ACT has established strategic partnerships with hardware and software manufacturers providing Ada 95 capabilities. In the former category is ACT's relationship with Silicon Graphics, Inc., whose new SGI Ada 95 product is based on GNAT. As of 1999, SGI sold over a billion dollars of Ada related software and equipment. ACT has relationships with several other hardware manufacturers such Compaq and Hewlett Packard. Commercial customers of ACT Europe include

Aerospatiale, Alenia, BNP, Boeing, British Aerospace, Canal+, CASA, Dasa, Ericsson, Hughes, Lockheed, TRW, etc."

To add some context to that quote above, Ada is a language designed specifically *for military, industrial and aerospace mission-critical and safety-critical systems where human life is routinely risked if software systems do not function correctly.* »

h) The challenge of change

It remains the case that moving from an environment of proprietary software to one of free software imposes costs on firms and organizations. Some do it anyways! A few well-documented examples, chosen from many, will suffice to illustrate the point.

In 2003, the municipal government of Munich[12] switched its entire computer system from MS Windows to the open source operating system Linux. This amounted to 14,000 computers in the public administration running a new operating system and also being equipped with other open source applications. The prime objective of this change was to deploy an information technology that would bolster commercial and technological flexibility at a low cost. Many reasons underlay this choice:

(i) independence: to ensure that the City possessed greater independence, in terms of IT, by breaking its reliance on a single solution or a single seller;

(ii) freedom: to create greater freedom of choice, leaving the decision of the best time to renew software with individual users;

[12] References: Blau John, « Munich Chooses "LiMux" Over Microsoft », 02/2005.
Silicon.com. « Norwegian city government switches to open source », 02/2005.
Desktoplinux.com. « Munich goes with Open Source Software », 02/2005.
CRN. « Munich Approves Changeover from Microsoft to Linux ».
USA Today. « Linux took on Microsoft, and won big in Munich »
Galli Peter. « Microsoft Loses Munich Deal to Linux ».

(iii) cost reduction: the low cost of deploying Linux, combined with the fact that updates cost nothing, eliminates the costs of renewing the licence with Microsoft;

(iv) efficiency: greater process efficiency;

(v) consolidation of the workload;

(vi) a better return to investments in the long term.

In November 2004, Germany's national railway company[13] equipped its 300 servers with open source technology. This migration is ongoing, as Lotus Notes is replaced by open source collaborative software. The switchover will affect a total of 55,000 users and is the most far-reaching migration project of the type. Also, by the end of the year, the free and open source operating system Linux will be running all databases, application servers, Web servers, and network infrastructures. The company hopes that this change will provide an information technology that is efficient and economical. Among the reasons given for this choice, we find:

(i) cost: the firm expects to save 50 per cent by adopting this open source technology;

(ii) flexibility: Linux is independent of any particular software publisher, making it more flexible;

(iii) strategy: Linux is deemed a vital component of the company's corporate strategy.

Sherwin-Williams[14] chose to replace its Unix operating system with Linux. This open source operating system will be installed in the firm's 2500 stores in the United States, Canada, and Puerto Rico. By eliminating licence costs and royalties paid on each installation, the firm expects to save millions of dollars with Linux. It will be able to modify the source code to adapt it to its own uses. It foresees requiring flexible solutions to maintain its leadership role

[13] References: McCarthy K. « World's Largest Linux Migration Gets Major Boost »

IDABC. « German national railway moves 55 000 Notes users to Linux system »

Smith W. R. « German Federal Railways opt for Open Source to cut costs »

Millard E. « German Railways on Linux Track »

[14] References: BusinessWeek Online. « Giant Steps for a Software Upstart »

eWEEK.com .« Sherwin-Williams Picks an All-Blue Network »

Network World Fusion. « Sherwin-Williams picks IBM and Linux for stores » in a highly competitive business climate. According to analysts, Linux will provide access to higher quality technology at each point of sale, and ultimately allow a better provision of services to the clients.

4. Innovation, by design or evolutionary

4.1 Cathedrals or bazaars

A key issue to examine is as follows: What form of intellectual property rights is most conducive to innovation? As we mentioned in the introduction, there are two competing visions: open science, with the free circulation of ideas, or commercial innovation protected by intellectual property rights and/or trade secrecy. E. S. Raymond contrasts them with the image "cathedral or bazaar." Innovation under open science occurs in an incremental, decentralized, almost chaotic fashion. In a private context, innovation progresses in a planned and structured manner, guided by precise goals. As a rule of thumb, we do not believe that one approach is superior to the other, but rather that the relative benefits of these approaches depend on circumstances and that, globally, the two approaches are complementary and necessary. In this section, we examine the differences between open and proprietary innovative processes, and discuss how they may complement each other.

The protection afforded to private firms for their innovations ensures that they will continue to have an incentive to invest in applied research and in the creation of new products. The mission of private firms is to identify new products and services liable to generate value for consumers and to create and bring to market new products and services in the most efficient manner possible. Private firms should be encouraged to do this, and consequently must be able to capture a portion of the value that their innovations generate.

In a competitive environment, the pursuit of profit ensures the ongoing creation of innovations and an increase in consumer welfare. In general, governments are not particularly adept or effective in identifying "winners" in a given industry for purposes of industrial policy. It seems to us a poor basis for industrial policy for the government to seek to finance business innovation. It is better to leave it up to private firms to create and commercialize new products that satisfy the needs of consumers.

All of that being said, open science remains essential for the development of innovations. The twentieth century witnessed the emergence of a veritable scientific revolution. The numbers of researchers and of scientific journals and papers have grown exponentially in recent decades. Thanks to the Web, this growth was accompanied by greater access to knowledge. The concept of "open science," which is part of our lives, remains a vital engine of economic development.

In order to explain the exponential growth of scientific innovation, researchers have advanced the concept of "combinatorial innovation." Innovations do not only originate with the creation of new ideas, but also from combining and recombining pre-existing ideas and innovations. These new combinations, in turn, pave the way for yet another round of combinations and innovations. The goal of basic research is to create new methods and concepts which, in association with pre-existing knowledge, allow the development of innovations that were previously impossible. If Leonardo da Vinci, despite his genius, was unable to build his flying machine, this is not because he was less gifted than the Wright brothers, but rather because he did not have access to the internal combustion engine.

This principle of cross-fertilization of ideas is at the heart of the process by which innovations in pure and applied sciences (both natural and social) develop. The same principle underlies industrial and commercial development. Varian (2004) notes that the use of standards and interchangeable parts contributed to accelerating the innovation process in the United States at the end of the nineteenth century.[15] The patent system is also designed to encourage the revelation of trade secrets, allowing firms to take up existing innovations and combine them with others in order to create new products.

[15] "The attempts to develop interchangeable parts during the early nineteenth century are a good example of a technology revolution driven by combinatorial innovation. The standardization of design (at least in principle) of gears, pulleys, chains, cams, and other mechanical devices led to the development of the so-called "American system of manufacture" which started in the weapons manufacturing plants of New England but eventually led to a thriving industry in domestic appliances." - Varian [2004]

An effective system for protecting intellectual property rights and trade secrets is well suited for the creation of new proprietary systems and the invention of new commercial products that are costly to adapt to the needs of consumers and bring to market. However, this system is less well adapted to fostering innovations that rely on combining innovations from various sources. In fact, proprietary systems (especially in the world of software) tend to rely on trade secrets, so many of the innovations contained in them cannot be taken up and incorporated into other products. This effect is dampened when the innovating firm makes use of patents.

The role of the patent system is precisely to encourage the revelation of trade secrets in order to allow others to reuse the innovation. This makes it possible for a firm to take up the innovations of others and incorporate them into a new good or service. In the case of patents, a new problem appears, known as the "tragedy of the anticommons." When several patents, belonging to several firms, are combined to create a new product that is truly innovative, each patent holder may seek to appropriate some of the surplus generated by the new invention, ultimately leaving nothing to the firm that innovated and combined the patents. In practice, rather than serving to encourage the creation of innovations, patents can be used to block the entry of competitors into a market. In these situations, the patent system will impede, rather than foster, innovation.

Open science is ideal for encouraging the widespread diffusion of new ideas, facilitating the assembly and combination of small and diverse innovations, bringing together researchers, and ensuring a continual transfer and transformation of knowledge and know-how. Open science relies on the intrinsic motivation of innovators and their eagerness to contribute to the advancement of science and to promote their reputation in their academic, intellectual, and/or scientific communities. Open science is particularly appropriate for solving complicated and difficult problems that afford those who succeed glory and prestige. Since it is a challenge to raise funds for investments in generic public domain research, open science requires the input of public funds—one of the core roles of the government.

Thus, "open" and "proprietary" sciences are complementary, with each playing a role in the generation of new ideas and procedures and the creation of new goods and services.

4.2 Innovation in the area of software

In software, the two classes of innovation, open and proprietary, coexist. The proprietary system made possible the rise of giants like Microsoft and SAP, while the open system allowed the creation of operating systems like Apache and Linux, office software like OpenOffice, and the development of open standards like HTML, (hyper text markup language) and HTTP (hyper text transfer protocol). You don't need to be psychic to foresee that the upcoming years will witness major innovations and transformations in the area of software and the Internet.

At first blush, we may be inclined to think that a system of innovation based on proprietary software is better adapted to the development of software. Software development is not a basic, pre-competitive research activity, but rather the culmination of a will to create and disseminate a commercial product. This puts it more into the class of industrial development. In this context, an approach based on free software should be unable to compete with proprietary software. This is the paradox: The approach appears to work and be an effective way to create value and generate significant innovation in the field of software. This paradox is attributable to some specific features of software and to the recent evolution of information and communications technologies.

The advent of Internet technologies contributed to accelerating the innovative process, especially where it was based on the open science model. The Internet has allowed a much wider dissemination of ideas. Moreover, Internet standards allow the easy combination and reuse of different contributions and innovations to create new services and products. The genius of HTML and HTTP is in how they allow pages created around the world to be linked and assembled. Blogs and wikis are tools whose role is to share knowledge. Founded in 2001, the free encyclopaedia Wikipedia has become a vast reference work in over 30 languages with nearly a million articles in English alone. This colossal documentary resource is the fruit of voluntary contributions and is freely accessible on the Web. This type of achievement is made possible by the fact that wiki Web technology allows the assembly, classification, and management of contributions from a large number of sources.

The existence of blogs and wikis reflects the impact of new Web technologies, but also a powerful social phenomenon that is specific to Internet culture: the Agora effect. The Internet allows all sorts of communities to spring up, often rapidly and spontaneously. Internet groups exchange advice and information. Role playing games of over 10,000 players exist on the Internet, in which each participant assumes an identity in virtual reality. Participants spend hours conquering new territories, negotiating treaties, concluding alliances, and climbing the social ladder. Clients of Amazon submit comments on the books they have read. Parents share their experiences with others. The Internet has become a veritable locus of social exchanges and interactions. The development of free software is partly based on the spontaneous formation of these communities. The surprising success of Linus Torvalds, the instigator of Linux, is attributable to the force of the community he succeeded in creating around his project. Today the Linux support community has access to considerable human and financial resources. The Agora effect is a new and powerful phenomenon, and certain private firms have understood the value of tapping into it.

Recent developments in software engineering allow existing software components to be reused and recombined. The dynamic nature of the business world imposes a constant acceleration on the development of computer applications for e-commerce transactions, be they inter- or intra-firm. One very promising avenue for increasing this speed is to reuse predefined components that are easily assembled and adapted to a specific context. This reuse can apply to generic elements, i.e. those that are common to various situations, or templates, i.e. solutions proven to work in similar situations. To be effective, it must allow the unique characteristics of each context to be captured and exploited.

A number of the technologies having appeared in recent years allow increased reuse of source code: UML (Unified Modeling Language), ebXML, (Electronic Business using eXtensible Markup Language) and MDA (Model Driven Architecture). Simultaneously, some professional organizations have created catalogues of generic processes that are specific to their industry, in the hope of fostering exchanges between firms. These recent developments ultimately allow templates describing business processes for adapted and reusable computerized tools to be created. Web services rely on standards and technologies that allow computers to dynamically assemble the information and resources available on the Web as needed.

Thus, the very nature of software, along with recent developments in Web technologies and software engineering, create an environment on a model that is propitious to innovation based on open source code and the instantaneous sharing of knowledge.

4.3 Why contribute to a free and open source software project?

In terms of economic analysis, one key and paradoxical issue involves understanding why individuals and firms invest time and resources to support free software projects. Contributions to such a project can come from three sources: (i) individuals, i.e. independent developers seeking to contribute to the creation of new solutions on a voluntary basis and thus make a name for themselves within the community; (ii) software publishers who opt to develop their projects as free software because that strategy has benefits from a developmental perspective; and (iii) firms using the software and seeking to improve it for their in-house uses, judging that they are better off sharing their contributions with the community, at least partly to be able to capitalize on the forking effects discussed earlier.

Lerner and Tirole (2000) attempt to explain why individuals agree to contribute to free software projects. By participating in free software projects, individuals have the opportunity to learn, become known in the community, and eventually sell their (by then) established expertise in the fields of integration or customization. Giant corporations like Sun, HP, and IBM have invested substantial resources in funding free software projects. In and of itself, this may appear surprising. Why would for-profit firms commit financial and human resources to the creation of products to be distributed free of charge to its clientele, and further leave these products open to examination by competitors with no protection under trade secrecy? The firms supporting free and open source software projects must assess whether it is to their advantage (less expensive and more efficient) to use an open approach for creating and developing software than to use an approach based on trade secrecy. Clearly, there must be some economic interest in doing so. This interest may assume various forms.

1) Products created as "free software" are complementary to proprietary products sold by the same companies. Thus, by making free software solutions available on the market, these firms increase the value of their other products and, in so doing, increase the profits derived from them. Firms having initiated free software projects render the investment profitable by offering support, training, etc. These services and activities can be very lucrative.

2) Software has become very complex. The hope of being able to plan and control everything is clearly delusional. A more incremental approach, based on trial and error, is undoubtedly wiser, less risky, and more profitable. Through use of this open approach, these firms can harness the voluntary and free support of many external collaborators and dip into a broader pool of expertise. This can accelerate the innovation process and the identification of bugs and requirements.

3) An open approach may also boost the motivation of in-house developers by stimulating their competition with external developers while allowing them to enhance their reputation externally. In making the source code available to examination by other developers, the firm may rapidly learn whether the free software project elicits the interest of the community and whether the competencies of in-house developers are recognized by the community. For in-house developers, a free software project may provide the opportunity to acquire external recognition and increase their value on the market. In both cases, greater transparency tends to alleviate the problem of moral hazard between employers and developers: The former are ensured of higher quality

and greater effort in the work of the latter, while workers obtain the assurance that it is in their employers' interest to pay them at fair market value.

4) In the area of software, where forking and network effects are very pronounced, it is difficult, if not impossible, to replace competing proprietary solutions that are widely used. Only a free software strategy appears adequate to allow Sun, HP, or IBM to mount a credible competitive challenge to Microsoft.

In the framework of a free software development project, it may be beneficial for a firm that is using this type of software, and having the required competencies and resources, to contribute to its development and to submit the improvements to the community. In contrast to what one might be inclined to believe, free software is not "free." The appropriate formula is: "Free software is only free if your time is free!" In fact, free software solutions are frequently less user-friendly than proprietary solutions. Free software developers are more interested in solving challenging technical problems than in documenting their work for laymen or in beautifying the interface. Customer support is often limited. As a rule of thumb, the users of free software need to have a basic expertise or resort to the help of professionals. However, if a firm has the required resources and expertise, a free software solution may be of some interest. The first advantage is that free software is designed to be open-ended, flexible, modular, and easily adaptable. Except in the case of office software, which is relatively standardized, business applications need to be adapted to the specific needs of the firm. Consequently, a company seeking to implement an IT system must invest resources to adapt it to its needs. Under some circumstances, it will need to add to the source code to effect this customization.

5. Economic cost-benefit analysis applied to free and open source software

Governments often have a hard time articulating a clear industrial policy, tending to assign it goals that are frequently poorly defined, too numerous, and contradictory. They typically wish to support job creation, the regions and also metropolitan centres, local business, the creation of industrial clusters, etc. This confusion regarding the objectives, along with the proliferation of associated government programs, undercuts the effectiveness of efforts to define an industrial policy.

The primary task of an economic system is the creation of value. The Canadian economy must be able to create value. In the area of software, value creation corresponds to an increased use of software that is useful, powerful, fit for the task, and inexpensive to produce. Thus, supporting a local software industry that creates expensive and ineffectual solutions is not good policy—diverting scarce and valuable resources into activities with limited, if any, productivity.

5.1 Economic spin-offs

Governments are typically concerned about economic spin-offs, a concept that is often poorly understood and misused. The term "economic spin-offs" is often misguidedly used to refer to project costs, i.e. the expenses incurred to implement a project and that are recorded as income by service providers.

The objective of economic cost-benefit analysis when evaluating a project or a government policy is to compute the true economic spin-offs. These correspond to the real benefits generated by a project. These do not consist of income transfers, but rather the actual creation of value. If a sum of money is simply transferred from one economic agent to another, then the profit accruing to the one is identical to the loss incurred by the other, and no net benefit can be recorded. If, however, it results in the availability of a product or service that is inexpensive to produce and procures substantial benefits to its consumers, then value has been created.

In general, competitive market mechanisms allow economically beneficial activities to occur; if it is possible to make a profit by selling a service or product that is desired by consumers, then this will happen. The government is justified in becoming involved in funding part of this activity if and only if it generates externalities, i.e. real benefits that the project's promoter cannot exploit or appropriate and that cannot be taken into account by the unit or agent underwriting the project.

In this section we will seek to explore the benefits and disadvantages of using one or the other type of software licence by applying the rules of economic cost-benefit analysis. We shall draw some key conclusions.

5.2 Granting an exclusive right to the commercialization of software to one firm and allowing it to block access to the source code destroys value.

Granting an exclusive right to the commercialization of software to one firm and allowing it to block access to the source code is tantamount to transferring value to that firm. This claim has an economic value corresponding to the economic rent that the firm becomes able to extract on the market in the form of profits from the sale of licences and services. So why do we say that granting this right destroys value? Because the economic rent that the firm receives owing to its exclusive right springs from the fact that the price of the licence exceeds the amount it could have charged had the software been open source. Thus, the profit earned by the firm owning the code is, at most, equal to the users' loss, and generally it is considerably below this loss.

When software is developed and its source code made freely available, substantial benefits are generated. The source code that has been created and shared has a value from which others can benefit. Access to the source code allows them to meet certain needs. Furthermore, access to the source code allows in-house or external developers to see the structure of the code, to learn how it works, and to extrapolate how similar projects might be created. Overall, these benefits will tend to exceed the profits that the firm owning the code could hope to receive from exclusively commercializing the rights. In order to capture this rent, the firm prices its product at above the marginal cost of production (which is often nil), resulting in under-usage of the code. Furthermore, by restricting to a single firm the right and responsibility to promote the intellectual property incorporated in the code, we are probably eliminating some promising opportunities. Making the code proprietary reduces its value.

5.3 Allowing a firm to take up open code in order to improve it and eventually commercialize it under a proprietary licence creates value.

The protection of intellectual property rights is a necessary evil. It is an "evil" because it confers monopoly power on owners in the exercise of their property rights. As we have just seen, this creates economic inefficiency by restricting competition and access to the product. However, it is necessary because the protection of intellectual property rights creates powerful incentives for the creation of new ideas.

A firm may wish to modify open source code and commercialize the result. Giving this firm the opportunity to make improvements and commercialize the result allows value to be created. Opponents of this view assert that this is equivalent to letting firms make profits from appropriating a public good. This argument is erroneous. The source code remains free and accessible. Those who want to can reuse the open source code, adapt it to their needs, and choose whether or not to submit the modifications to the community. Furthermore, the economic rent that the firm will be able to appropriate from commercializing its proprietary code will depend on the extent to which it has added value to the initial code. To the extent that users retain access to the initial source code, the firm will not be able to price its product higher than the value of the marginal improvements it has contributed.

Barring a firm from taking up open code in order to improve and possibly commercialize it under a proprietary licence limits the scope for innovation. Clearly, it would be preferable if all innovations contributed to open source code could revert to the free software community. In reality, however, some innovations may be costly, and certain firms may decide to only devote resources if they are able to commercialize the results of their investments under a proprietary licence. Stakeholders must be given freedom of choice, and then competition will determine what type of licence is most appropriate for stimulating innovation under the circumstances.

5.4 A policy designed to systematically support proprietary software as a way of boosting the software industry is not justified on the basis of economic calculus.

Some have suggested that software consumers require a prosperous IT services industry to supply the related support and services and that, in the absence of exclusive rights to commercialize, software publishers cannot be financially viable. This argument is erroneous. If software users need support and the supply of complementary services to be able to make use of open source code, an economically viable industry supplying these services will arise. Since the source code is open, no artificial barrier would block entry into this service industry, and competition would compel it to be efficient.

In the context of projects based on open source software, expenditures on licence fees are reduced but outlays on services, customization, and updates will undoubtedly remain high. A project of implementing free software bolsters the local industry as much as a project using proprietary licences, the difference being that licence fees are lower and consulting fees are higher.

The primary task of an economic system is the creation of value. The Canadian economy must be able to create value. In the area of software, value creation corresponds to increased use of software that is useful, powerful, fit for the task, and inexpensive to produce. Thus, supporting a local software industry that creates expensive and ineffectual solutions is not good policy—such a policy diverts scarce and valuable resources into activities with limited, if any, productivity. Hundreds of millions of dollars invested in huge IT projects that failed could have gone elsewhere.

Government policies targeting the software industry must first and foremost support and encourage competition and innovation. These two objectives are linked: Competition is one of the key elements of any policy to support innovation. To survive in a competitive world, firms must innovate and constantly seek to satisfy the needs of their clientele. Competition must also allow consumers to capture some share of the value created by innovations.

To successfully support innovation, market forces must be harnessed as much as possible. Governments must remain neutral with regard to the use of free software outside of their immediate sphere of influence and their own needs as users. As a rule of thumb, governments are poor judges in the matter of picking winners.

Conversely, as users and consumers of technology, governments can and must protect their own interests and ensure that the retained solutions most adequately satisfy their own needs. They must avoid biasing the competitive process by supporting any solution (domestic or foreign) on any basis other than the product's cost and quality. Free competition must reward the most meritorious firms. Similarly, computer services contracts issued by the government must not be hidden subsidies to a local or foreign industry or firm.

The government must recognize the right of innovating firms to protect their intellectual property. This does not mean, however, that the government should be prepared to pay more for proprietary software than for free software of equivalent quality, on the premise that it should promote the development of intellectual property. The government must buy the product most suited to its needs at the best price.

5.5 The criteria applied to the choice of software solutions must account for externalities and long-term impacts.

The decisions of public administrations are largely driven by the budgetary constraints that are imposed on them. Too often, decisions are made that do not account for externalities or long-term impacts. As a result, some decisions made by public administrations run counter to the economic optimum.

A widespread practice in public administrations is to assign all intellectual property rights over the created code to firms that develop applications for the government. This practice allows the administrative unit to reduce its purchase price. In our opinion, this policy is ill conceived from an economic perspective. When government bodies seek to reduce their software costs and compensate firms by giving them monopoly powers in the exploitation of work conducted to fulfill the government's needs, this is an economically inefficient method of financing these contracts. In particular, by assigning monopoly power over the code created, the government is undercutting future competition and could end up paying more for updates or adjustments to the software that it may eventually need.

Another distortion attributable to management by annual "budgetary allowances" in public administrations is that decisions are made on the basis of very short-term objectives. A key criterion that governments should use when evaluating software solutions is their capacity to evolve over time. Software is unique: It must be able to adapt to new technologies and standards and evolve to satisfy changing demands. The purchase of software that will require costly updates and adaptations in the near future is economically disastrous. The result is the purchase of software that is less expensive in the short run, but also less adaptable. In the medium and long term, these practices may prove very costly. Unfortunately, public administrations tend to live with short-term budgets and set short-term goals.

The level of flexibility is a criterion that must be spelled out when evaluating software for IT solutions. In particular, the government must avoid lock-in situations, in which the buyer of computer services becomes captive to the supplier. The government must avoid signing service contracts that provide its suppliers with *de facto* market power in the provision of future services.

Thus, a cornerstone of the government's strategy should be to rely on internationally recognized open standards. Adopting open standards ensures that several suppliers will be able to provide the required support. If all government documents complied with OASIS (Organization for the Advancement of Structured Information Standards) open standards,[16] they could be read, interpreted, and modified by any software designed to work with those standards. One area of application is the protocol Open Document Format for Office

[16] OASIS (http://www.oasis-open.org/who/) is a non-profit international organization striving to promote the development, convergence, and adoption of e-commerce standards. OASIS is a consortium comprising over 600 individual member organizations from more than 100 countries.

Applications: OpenDocument.[17] Under this open standard, any software publisher can write a utility to create, modify, or interpret documents. This fosters competition between office software developers and allows various users (including individuals) to select software that meets their needs while encouraging competition, innovation, and the creation of new products by local firms.

[17] The OpenDocument standard seeks to define an open standard for documents generated by office productivity software.

6. Recommendations

In this last section we summarize our conclusions and offer six recommendations.

1) **The government should strongly encourage the use of open standards that are internationally recognized. It should avoid adopting proprietary solutions that limit competition and leave it dependent on a single supplier.**

Interoperability of information systems will be one of the major challenges in coming years. One solution envisaged for allowing a large number of actors to share information is to construct centralized systems built on shared databases. This approach has proven excessively expensive and often unachievable. An alternative that has proven much less costly consists of allowing several systems to exchange shared information. This requires that different information systems are capable of "talking to each other" and automatically exchanging the required information. For this to work, computers must understand each other and share communications protocols, and the documents must be built on universal semantic structures. To satisfy these requirements, considerable effort has been invested in creating open standards and large international consortia have devoted significant resources to identifying such standards. The government must not isolate itself from this movement—it must rather be a participant.

2) **To the extent possible, and in the absence of compelling reasons, the government must avoid assigning exclusive commercialization rights for software developed in response to government needs. The government should encourage open source licences.**

Granting an exclusive right to the commercialization of software to one firm and allowing it to block access to the source code destroys value. The protection of intellectual property rights should serve to compensate innovation, and nothing else. Software specifically commissioned to satisfy needs defined by government departments should not be afforded protection under the intellectual property rights regime. Assigning intellectual property rights to an accomplishment requiring little, if any, true innovation does not serve the goals underlying intellectual property rights protection, but only awards rents and market power to a firm—for no good reason. The government should insist that software created for its needs be built around open standards and released to the open source software community.

3) The government should promote the use of open source licences, while still allowing firms to incorporate the code into commercial products (BSD licences).

Allowing a firm to take up open source code in order to improve it and eventually commercialize it under a proprietary licence creates value. It must be recognized that there are several sources of innovation in the field of software. Allowing open source code to be taken up and transformed for commercial purposes does not in any way detract from the initial source code, but it sets the stage for useful innovations. Thus, licences that allow for the reuse of code should be encouraged as much as possible.

4) When the government uses and adapts free software, it should return the software containing the improvements and adaptations to the community.

There are certain benefits to submitting the modifications made by a government body back to the free software community. This primarily allows (i) easier and quicker detection of any errors made, and (ii) avoiding the phenomenon of "code fork," i.e. adaptations becoming incompatible with the evolution of the core source code. Thus, except in cases in which the security of the system or the protection of confidential data make it unadvisable, the government should submit its contributions to the source code to the community.

5) The government should adapt its administrative practices to explicitly incorporate calculations of long-term impacts and externalities in its software choice criteria.

Administrative practices are not always well adapted to the complexity of software selection issues. Software is a complicated product that must evolve. There are important network effects associated with the use of software. If all administrative units were to choose IT solutions without accounting for externalities and long-term effects, the process would be inefficient. The challenge of ensuring that externalities and dynamic effects are accounted for is considerable. For this, the government must articulate a clear vision.

6) **The government must support the development of a network of expertise in free and open source software, one of whose roles would be to promote free software.**

Proprietary software publishers have an interest in investing a lot of resources in promoting their software. Consequently, it is easy to find experts who are able to explain the virtues of proprietary software to us. Unfortunately, there is no, or virtually no, counterpart in the domain of free software. Thus, it is difficult for public administrations to access expertise in the area of free software with the competence to help them select IT solutions. In order to shed some light, the government must stimulate the creation of such expertise.

Bibliography

Australian Government Information Management Office (2005). “A Guide to Open Source Software for Australian Government Agencies. Developing and Executing an ICT Sourcing strategy”. http://www.sourceit.gov.au/__data/assets/pdf_file/42065/A_Guide_to_Open_Source_Software.pdf

Blanas, George (2003). “2SOS-ware DEVILS, [Strategic Open Software DEVelopment ILlnesseS]”, Department of Project Management, TEI of Larissa, Greece.

Bessen, James (2004). “Open Source Software: Free Provision of Complex Public Goods”, ResearchOnInnovation.org. http://www.researchoninnovation.org/opensrc.pdf

Dalle J-M, David P.A., Ghosh R.A. and W.E Steinmueller (2005). “Advancing Economic Research on the Free and Open Source Software Mode of Production”, forthcoming in *How Open is the Future?*, Edited by Marleen Wynants and Jan Cornelis, Vrjie Universiteit Brussels (VUB) Press, Brussels.

Danish Board of Technology (2002). “Open-source software in e-government”, Danish Board of Technology. http://www.tekno.dk/pdf/projekter/p03_opensource_paper_english.pdf

Ghosh, R.A. (2002), “Free/Libre and Open Source Software (FLOSS): Survey and Study”, International Institute of Infonomics, University of Maastricht, The Netherlands; Berlecon Research GmbH, Berlin, Germany (http://flossproject.org/report/index.htm)

Hawkins, R.E. “The Economics of Open Source Software for a Competitive Firm, Why give it away for free?”, *Netnomics* (forthcoming)

Johnson, J.P. (2002). “Open Source Software: Private Provision of a Public Good,” *Journal of Economics and Management Strategy,* v. 11, no. 4, pp. 637-62.

Johnson, J.P. (2001). “Economics of Open Source”, http://opensource.mit.edu/papers/johnsonopensource.pdf

Kuan J. (2001). “Open Source Software as Consumer Integration into Production”, Haas School of Business, University of California-Berkeley. http://papers.ssrn.com/sol3/papers.cfm?abstract_id=259648

Kenwood, C.A. 2001. “A Business Case Study of Open Source Software,” Report #MP01B0000048, MITRE Corporation: Bedford, MA. http://www.mitre.org/support/papers/tech_papers_01/kenwood_software/

Lerner, J. and J. Tirole (2002). “Some Simple Economics of Open Source”, *Journal of Industrial Economics*, Vol. 50, no. 2 , 197-234.

Lerner, J. and J. Tirole (2004). “The Economics of Technology Sharing: Open Source and Beyond”, Working Paper 10956, NBER Working paper series, http://www.nber.org/papers/w10956.

MITRE Corporation (2003). “Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense”, Prepared for The Defense Information Systems Agency (DISA). http://www.egovos.org/rawmedia_repository/588347ad_c97c_48b9_a63d_821cb0e8422d?/document.pdf

Raymond, E.S. (2001). “The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary”, O'Reilly & Associates, Farnham, UK Feb 2001, ISBN 0-596-00108-8.

Schmidt, K.M. and M. Schnitzer, (2002). “Public Subsidies for Open Source? Some Economic Policy Issues of the Software Market”, University of Munich, CEPR and CESifo.