CPB Document

No 181

March 2009

Competition, innovation and intellectual property rights in software markets

Michiel Bijlsma, Paul de Bijl and Viktória Kocsis

CPB Netherlands Bureau for Economic Policy Analysis
Van Stolkweg 14
P.O. Box 80510
2508 GM The Hague, the Netherlands

Telephone +31 70 338 33 80
Telefax +31 70 338 33 50
Internet www.cpb.nl

ISBN 978-90-5833-397-1

Abstract in English

This study analyzes under which circumstances it may be desirable for the government to stimulate open source software as a response to market failures in software markets. To consider whether policy intervention can increase dynamic efficiency, we discuss the differences between proprietary software and open source software with respect to the incentives to innovate and market failures that may occur. The document proposes guidelines to determine which types of policy intervention may be suitable. Our most important finding is that directly stimulating open source software, e.g. by acting as a lead customer, can improve dynamic efficiency if (i) there is a serious customer lock-in problem, while (ii) to develop the software, there is no need to purchase specific, complementary inputs at a substantial cost, and (iii) follow-on innovations are socially valuable but there are impediments to contractual agreements between developers that aim at realizing such innovations.

Key words: Software markets, Intellectual property rights, Open source software, Public policy
JEL code: L17, L52, L86, O34

Abstract in Dutch

Deze studie analyseert wanneer het wenselijk is dat de overheid open source software stimuleert in reactie op marktfalens in softwaremarkten. Om te beoordelen of beleidsinterventie de dynamische efficiëntie kan verhogen, bespreken we de verschillen tussen 'proprietary' en open source software met betrekking tot de prikkels om te innoveren en marktfalens die kunnen optreden. Deze studie stelt richtlijnen voor om te bepalen welke vormen van ingrijpen geschikt kunnen zijn. De belangrijkste bevinding is dat het direct stimuleren van open source software, bijvoorbeeld door als aanjagende klant op te treden, de dynamische efficiëntie kan vergroten als (i) de leveranciersafhankelijkheid groot is, terwijl (ii) er in de ontwikkelingsfase geen specifieke, complementaire input tegen substantiële kosten ingekocht hoeft te worden, en (ii) vervolginnovaties sociaal wenselijk zijn maar er beperkingen zijn om die door middel van contracteren tussen ontwikkelaars tot stand te laten komen.

Steekwoorden: Softwaremarkten, Intellectuele eigendomsrechten, Open source software, Overheidsbeleid

Contents

Preface		7
Summary		9
Nederlandse samenvatting (Summary in Dutch)		13
1	Introduction	17
2	Definitions and delineation	19
3	Characteristics of open source and proprietary software	23
3.1	Incentives to participate in open source software development	23
3.2	Market failures related to software	29
3.3	Effects of incentives and market failures on welfare	34
4	The potential role of the government	37
4.1	Policy implications	37
4.2	Government failures	46
4.3	Summary	47
5	Conclusions and discussion	49
References		51

Preface

The Dutch parliament and government intend to stimulate efficient electronic information exchange, and to do so, see an important role for innovations, particularly aiming at interoperability and competition in software markets. It has therefore proposed various measures to stimulate the adoption of open standards and open source software in the public and semi-public sectors. On the request of parliament in 2007, the Ministry of Economic Affairs published the Action plan *Nederland Open in Verbinding* on how to prepare and implement these ideas. The Ministry of Economic Affairs has asked CPB to analyse the economic effects of promoting open source software on competition and innovation. Based on existing insights from the economics literature, this study analyzes policy options in the light of software market characteristics, such as the incentives to innovate under different software licensing regimes and market failures that may occur in these markets. The document proposes guidelines to determine under which circumstances various types of policy intervention may be desirable to improve dynamic efficiency.

This document was written by Michiel Bijlsma, Paul de Bijl and Viktória Kocsis. Viktória Kocsis acted as project leader. The authors gratefully acknowledge the extensive comments by and the discussions with the advisory committee of the Ministry of Economic Affairs. The committee consisted of Jan Julianus, who had initiated the contact with CPB that led to this project and chaired the meetings, Margreet Groenenboom, Yvonne van Santen, Hendrik Brandwijk and Tjade Stroband. The authors also wish to thank Rob Aalbers (CPB), Jan Boone (UvT), Madeleine de Cock Buning (CIER/UU), Casper van Ewijk (CPB), Marco Haan (RUG), Berend Hasselman (CPB), Pierre Larouche (TILEC), Hans Lunsing (CPB), Bas Straathof (CPB), Erwin Zijleman (CPB), and participants of our internal seminars for useful suggestions and discussion. The responsibility for the content and the conclusions of this report rests entirely with CPB.

Coen Teulings
Director CPB

Summary

The Ministry of Economic Affairs has recently issued the Action Plan *Nederland Open in Verbinding* in which it aims at stimulating the use of open source software (OSS) in order to increase interoperability and innovation and to reduce dependence on software suppliers. The Ministry has requested CPB to conduct research on the relationship between stimulating OSS and innovation and competition in software markets. This document addresses the following questions:

1. What are the incentives to innovate under different licensing regimes of software?
2. What are the market failures in software markets?
3. How do incentives and market failures affect welfare?
4. What can policy makers do to alleviate market failures and enhance welfare in software markets?

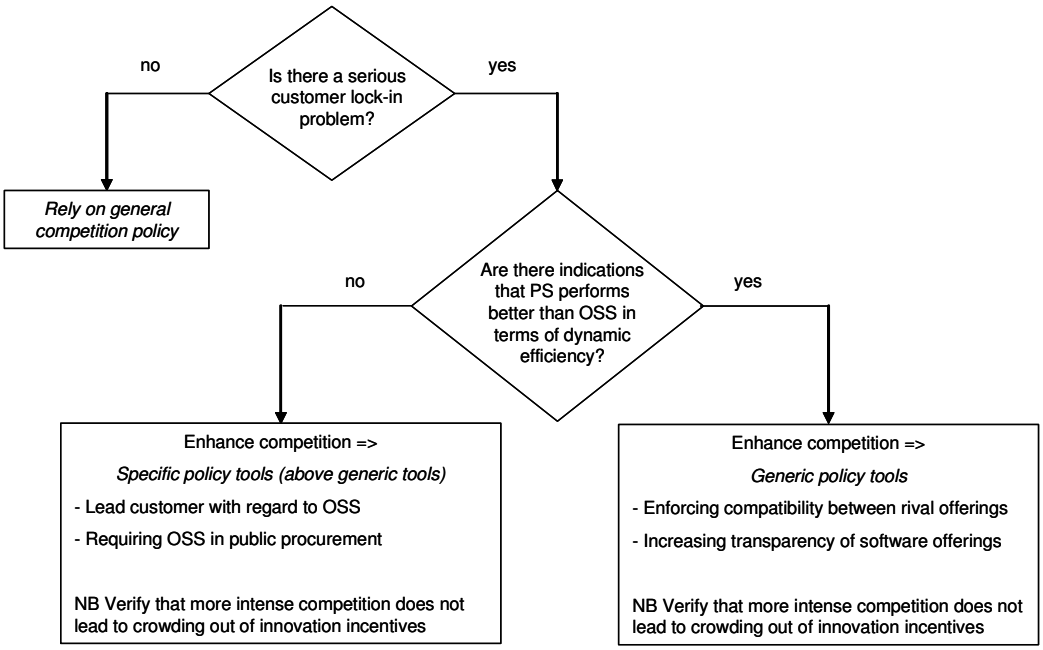
In Europe, software falls under a copyright regime. Different software types use intellectual property rights in different ways. In the case of proprietary software (PS), copyright protection is used to provide a temporal monopoly for the license owner in reproducing, distributing and developing derived works. This shields developers from direct imitation and makes it easier to have customers pay for the use of the product. OSS, on the other hand, is often available for free, furthermore, the license allows for access to the source code, such that others can adapt, improve and distribute this code without having to pay to the license holder. The license conditions of PS create prospects of profits from direct sales, stemming from the improved scope for charging a mark-up in the price. This supports, particularly in case of high development costs, the ex ante incentives to innovate. The conditions of OSS licenses limit the possibility to exercise market power, because everybody can use the source code to develop and distribute derived products. Therefore, the incentives to innovate must come from other sources, for instance from revenues of selling complementary products (e.g. customer support and advices, and complementary PS) and from the incentives of individual developers. These individual incentives can have an intrinsic nature (motivation coming from "within") or an extrinsic nature (e.g. signalling quality in the job market, self education and own use).

Both PS and OSS have advantages and disadvantages that relate to market failures in software markets. The most important market failures are market power and knowledge spillovers. Market power can exist due to network externalities (users benefit more from a larger network) and switching costs (material and perceived costs of switching to other software). In both cases customers get locked into the currently used product (there is "customer lock-in"). Another reason for market power relates to economies of scale which are particularly relevant when software development requires buying specific, complementary inputs in addition to programming skills. Consider for instance graphic designer skills in case of

computer games. Knowledge spillovers are particularly problematic when the prospect of free riding by others makes it less attractive to engage in R&D. In principle, PS licenses correct for this problem because they prevent free riding. However, a disadvantage of PS is that it can be too restrictive with regard to follow-on innovations. Hence, both software types have pros and cons: OSS directly encourages follow-on innovations, but PS prevents that others can free ride on existing innovations (which is good for ex ante incentives).

This study suggests an outline to assess the different forms of market failures, with the aim to determine whether policy intervention is desirable. The following figure illustrates the central idea.

Figure 1 Decision tree for policy makers



The flow chart works as follows:

1. The first decision node indicates that if customer lock-in is *not* a serious problem – and hence there are no substantial entry barriers – generic competition policy is sufficient to deal with competition problems in software markets.
2. The second decision node indicates that if customer lock-in poses a serious problem in a particular software market, it is important to first determine whether it is desirable to aim policy specifically at OSS or to aim it generically at all software types. First, if there is a need to purchase specific, complementary skills at a substantial cost at the development stage, then PS offers better possibilities to recover these costs. Second, OSS stimulates follow-on innovations more directly, but PS prevents that others can free ride on existing innovations. Based on these criteria, one can determine which type of policy is most suitable to reduce the dependence on suppliers.

It is important to consider whether enhancing competition could go too far, in the sense that it would undermine innovation incentives. Theoretically this could happen if already at the beginning there is a high intensity of competition. However, when customer lock-in is present and incumbent suppliers experience little competitive pressure, it is likely that enhancing competition will stimulate innovation rather than reduce it.

The flow chart indicates when policy specifically aiming at OSS versus generic policy can be desirable to reduce entry barriers and increase long-term welfare. Examples of specific policy interventions are acting as a lead customer or giving a preference for OSS in public procurement. An example of generic policy intervention is imposing full compatibility so that files can be exchanged seamlessly between applications of different suppliers.

Because of the risk of government failure, policy instruments have to be used carefully. The government has an information disadvantage compared to market participants; therefore it is generally advisable to abstain from policies that select a winner in advance. Also, specific policy tools often invite lobbying to influence policy in an early stage. Due to incomplete information, the design of policy intervention can be distorted, apart from inefficiencies related to lobbying activities. Note that the more generic policy is, the less these problems play a role.

Nederlandse samenvatting (Summary in Dutch)

Met het Actieplan *Nederland Open in Verbinding* beoogt het ministerie van Economische Zaken het gebruik van open source software (OSS) te stimuleren, ten behoeve van interoperabiliteit, innovatie en terugdringen van afhankelijkheid van softwareleveranciers. Het ministerie heeft het CPB verzocht onderzoek te doen naar de relatie tussen het stimuleren van OSS en innovatie en concurrentie in de softwaremarkten. Dit document adresseert daarom de volgende vragen:

1. Hoe beïnvloeden verschillende licentiemodellen de prikkels om software te ontwikkelen?
2. Wat zijn de marktfalens in softwaremarkten?
3. Hoe wordt welvaart beïnvloed door innovatieprikkels en marktfalens?
4. Wat kunnen beleidsmakers doen om softwaremarkten beter te laten werken?

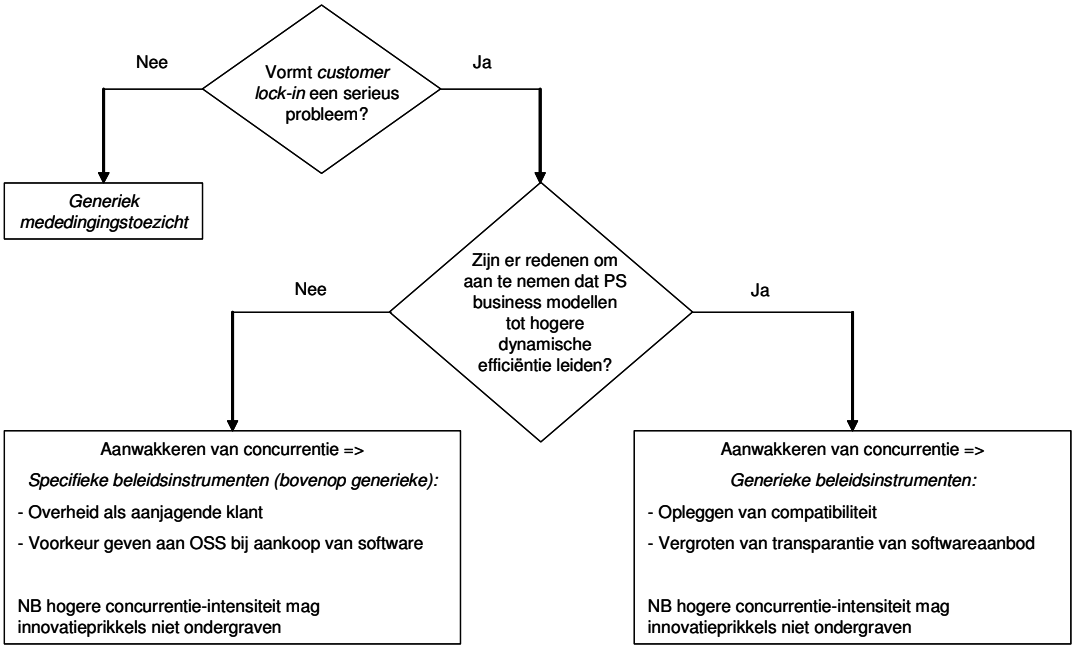
Op software rusten in Europa intellectuele eigendomsrechten in de vorm van auteursrechten. Verschillende softwarevormen gebruiken auteursrechten op een verschillende manier. Ten eerste is er *proprietary software* (PS). Bij PS maken auteursrechten een tijdelijk monopolie mogelijk ten aanzien van reproductie, distributie en ontwikkeling van afgeleide werken. Dit beschermt ontwikkelaars tegen directe imitatie en stelt hen daardoor in staat consumenten of bedrijven te laten betalen voor gebruik van het product. Ten tweede is OSS, dat veelal gratis beschikbaar is en biedt de licentie toegang tot de broncode, zodanig dat anderen deze mogen wijzigen, verbeteren en distribueren zonder daarvoor te hoeven betalen aan de houder van het auteursrecht. De licentievoorwaarden van PS creëren vooruitzichten op winst uit directe verkopen vanwege de mogelijkheid om een mark-up in de prijs op te nemen. Dit waarborgt, vooral bij forse ontwikkelkosten, de ex ante prikkels om te innoveren. De licentievoorwaarden van OSS beperken de mogelijkheid marktmacht uit te oefenen, omdat iedereen op basis van de broncode zijn eigen variant kan ontwikkelen en verspreiden. Innovatieprikkels komen dus uit een andere hoek, zoals de winst door verkoop van complementaire producten (bijv. ondersteunende diensten, advies, en complementaire PS) en prikkels van individuele ontwikkelaars. Dat kunnen zowel intrinsieke prikkels zijn (motivatie uit eigen beweging) als extrinsieke (bijvoorbeeld kwaliteiten demonstreren op de arbeidsmarkt, leereffecten, eigen gebruik).

Zowel PS als OSS heeft voor- en nadelen die gerelateerd zijn aan de marktfalens die kunnen optreden in softwaremarkten. De belangrijkste zijn marktmacht en kennisspillovers. Marktmacht kan ten eerste ontstaan door netwerkeffecten (gebruikers ontlenen meer nut naarmate er een groter gebruikersnetwerk is) en overstapkosten (materiële of immateriële kosten van overschakelen op andere software). In beide gevallen kan leveranciersafhankelijkheid (*customer lock-in*) ontstaan. Een tweede oorzaak betreft schaaleffecten die vooral belangrijk worden wanneer er specifieke, complementaire expertise

ingekocht dient te worden ten behoeve van de ontwikkeling van software, aanvullend op de expertise van programmeurs. Denk bijvoorbeeld aan grafische expertise bij *games*. Kennisspillovers zijn vooral problematisch wanneer het vooruitzicht op *free riding* het minder aantrekkelijk maakt om R&D te verrichten. PS-licenties corrigeren daar in beginsel voor, omdat zij *free riding* tegengaan. Een nadeel echter is dat zij daarmee (te) beperkend kunnen zijn voor innovaties die voortbouwen op bestaande kennis. Beide softwaretypen hebben in dit opzicht dus voor- en nadelen: OSS kan sneller vervolginnovaties aanwakkeren, maar PS voorkomt dat anderen gratis meeliften op bestaande innovaties (wat gunstig is voor ex ante prikkels).

Deze studie doet een concreet voorstel om de verschillende vormen van mogelijk marktfalen in softwaremarkten te beoordelen, met als doel te bepalen of een beleidsinterventie wenselijk is. De volgende figuur illustreert het centrale idee.

Figuur 2 Beslissingsboom voor beleidsmakers



Het schema werkt als volgt:

1. De eerste stap geeft aan dat wanneer *customer lock-in* geen serieus probleem vormt – en er dus geen grote toetredingsbarrières zijn – generiek mededingingstoezicht volstaat om softwaremarkten goed te laten functioneren.
2. Wanneer de leveranciersafhankelijkheid in een bepaalde softwaremarkt groot is, is het belangrijk om eerst te bepalen of het wenselijk is om beleid specifiek te richten op OSS of om het generiek te richten op alle software. Ten eerste, wanneer in de ontwikkelingsfase dure expertise ingekocht moet worden, biedt PS betere mogelijkheden om de vooraf gemaakte kosten terug te verdienen. Ten tweede kan OSS kan sneller vervolginnovaties aanwakkeren, maar PS voorkomt dat anderen gratis meeliften op bestaande innovaties. Afhankelijk van deze

factoren kan bepaald worden welk type beleid het meest geschikt is om afhankelijkheid van leveranciers terug te dringen.

Het is belangrijk om na te gaan of het aanwakkeren van concurrentie via OSS innovatieprikkels zou kunnen schaden, wat in theorie zou kunnen optreden wanneer er al een hoge mate van concurrentie is om mee te beginnen. Echter, wanneer de markt op slot zit door *customer lock-in* en de gevestigde aanbieder daardoor weinig concurrentiedruk ervaart, is het waarschijnlijk dat meer concurrentie innovatie juist aanwakkert in plaats van verder afzwakt.

De beslissingsboom geeft aan wanneer beleid specifiek gericht op OSS, dan wel generiek beleid, wenselijk kan zijn om toetredingsbarrières te slechten en daarmee de welvaart op lange termijn te vergroten. Voorbeelden van specifiek beleid zijn dat de overheid *lead customer* (aanjagende klant) wordt of dat de overheid oplegt dat voor eigen inkoop voorkeur gegeven dient te worden aan OSS boven PS. Een voorbeeld van generiek beleid is het opleggen van volledige compatibiliteit, zodat bestanden naadloos uitgewisseld kunnen worden tussen applicaties van verschillende leveranciers.

Vanwege het risico van overheidsfalen is bij dergelijk beleid zorgvuldigheid geboden. De overheid heeft immers een informatieachterstand ten opzichte van marktpartijen, waardoor het in de regel verstandig is om geen beleid in gang te zetten dat bij voorbaat een winnaar selecteert. Ook geeft specifiek beleid vaak aanleiding tot lobbyen om het beleid in een vroeg stadium te beïnvloeden. Vanwege de informatieachterstand van de overheid kan het beleidsontwerpproces verstoord worden, nog los van de inefficiënties van het lobbyproces zelf. Dat speelt overigens minder naarmate beleid generieker is, dus minder gericht op een bepaald type software.

1 Introduction

In Europe, software falls under a copyright regime.[1] Copyrights, and more generally intellectual property rights (IPRs), grant developers a temporary, exclusive right to reproduce and distribute the underlying code, as well as to develop derived works.

Basically, there are two ways of licensing software: proprietary and open source licensing.[2] In the "traditional" case of proprietary software (PS), the underlying source code is typically not disclosed so that tinkering with the software by others cannot, and is not meant to, take place.[3] Consumers or firms can only use a particular program with the consent of the license owner. To obtain consent, users have to accept the license conditions and, in many cases, have to pay for the use of the software.[4] If the software is distributed or used without the consent of copyright owner, this constitutes a violation of copyright law.

The last decade or so a different licensing regime has emerged in software markets, which is commonly called open source licensing. In the case of open source software (OSS), the source code is made available to other developers under conditions specified by the license. OSS licenses typically allow others to modify and improve the source code, and distribute the software without payment to the license owner. Customers can then adapt, improve or distribute the source code and distribute the resulting (modified) program – under the conditions specified by the original open source license. In other words, OSS licenses typically use a watered down version of copyright, in the sense that the licensor claims less rights than could potentially be claimed by a copyright holder. Just as in the case of PS, open source licenses are enforced; not complying with their conditions violates copyright law.

The Dutch parliament and government intend to stimulate efficient electronic information exchange. The government sees an important role for innovations, particularly aiming at interoperability and competition in software markets. It has therefore proposed various measures to stimulate the adoption of open standards and open source software in the public and semi-public sectors. On the request of the parliament in 2007, the Ministry of Economic Affairs has published the Action plan *Nederland Open in Verbinding* on how to prepare and implement these ideas. The Ministry of Economic Affairs has asked CPB to analyse the economic effects of promoting open source software on competition and innovation:[5]

"17. Investigation into economic effects of open source software
The Bureau for Economic Policy Analysis (CPB) has been instructed to conduct an economic investigation into the relationship between the promotion of open source software and the

[1] In the US, patents are used to protect business processes that are translated into software. Think e.g. of process translated such as a "one-click" purchase format.
[2] Note that both licensing types are based on IPRs.
[3] Alternatively, the source code may be disclosed privately, under very stringent confidentiality obligations (the licensing terms may then be confidential as well).
[4] An example when consumers do not need to pay for the usage is freeware.
[5] Ministry of Economic Affairs (2007), p.23.

effects of this on innovation and competition in the ICT sector. There is increasing academic interest in the economic impact of the introduction of open source software. It is sensible to survey what these effects are, particularly on the functioning of software markets (for example to what extent customers' profit)."

In response to this request, we phrase the following questions in this study:

1. What are the incentives to innovate under different licensing regimes of software?
2. What are the market failures in software markets?
3. How do incentives and market failures affect welfare?
4. What can policy makers do to alleviate market failures and enhance welfare in software markets?

While the latter three questions speak for themselves, the first question needs some explanation. Innovation incentives receive special attention because of their importance for software markets. Moreover, IPRs are strongly connected with – and are meant to do so – innovation incentives. With regard to the other questions, when analyzing the role of the government, we assume it acts in the public interest, that is, it acts to enhance welfare. We study how stimulating OSS affects short-term welfare (static efficiency) and long-term welfare (dynamic efficiency).

There exist various other papers and publications that contain related or complementary policy analysis of the relative merits of OSS and PS. Some references are Knubben (2001), Schmidt and Schnitzer (2003) and Lee (2006). As the latter author states, the "policy considerations that inform government decisions are extremely complicated and sometimes interdependent" (p. 112). The complexity stems from the technological aspects in relationship to various forms and combinations of market failures that may occur in specific software markets – they are not uniform for the software market as a whole. Another study, Mendys-Kamphorst (2002), analyzes the impact of open source projects on competition in software markets.

For the sake of conciseness, this study exclusively focuses on open source versus proprietary licensing in software markets, and does not consider problems and issues related to open versus closed standards. Given the importance and complexity of standard setting issues, a separate study devoted to standards would be warranted. Nevertheless, it is important to note (also stressed by Lee, 2006) that policy specifically supporting OSS can have effects for which open standards fall short.

The remainder of this document is as follows. In chapter 2 we discuss the concepts relating to intellectual property right protection in software markets. Chapter 3 presents the incentives of participants, the market failures and their effects on market outcomes and welfare. The role of the government is discussed in chapter 4. Chapter 5 concludes the report.

2 Definitions and delineation

The role of intellectual property rights in the innovation process

Loosely speaking, intellectual property right (IPR) protection ensures an exclusive right to market a product during a given time period.[6] These exclusive rights effectively create the possibility of a monopoly position with regard to the product in question. Thus, *ex post* an innovation's investment costs may be (partially) covered by rents derived from this monopoly position. Two well-known forms of IPRs are patents and copyrights. Patents pertain to *ideas* and come into existence only after an application that was filed to the patent office has been granted. Copyrights aim at protecting the *embodiment* of ideas and apply automatically. From an economic perspective, an IPR regime is supposed to work as a mechanism to provide and protect the incentives to innovate. Such an incentive mechanism can take various forms. Hence, a fundamental issue concerns the optimal design of IPRs, in the sense that it minimizes the welfare loss from monopoly pricing, subject to the constraint that the incentives for innovation are not undermined.[7] In Europe, software falls under a copyright regime, therefore this study will focus on copyrights rather than patents.[8]

Arguably, copyright is not the perfect means to protect software, since it protects the software as a kind of work of art or literary work, and therefore might not offer sufficient protection against reverse engineering, that is, against extracting the underlying algorithm from the source code and reprogramming it into a 'separate work'.[9] Software developers may also rely on another option to protect the fruits of their intellectual efforts, namely trade secrets.[10]

Different types of license for software

Software is a general term for various programs that contain sequences of instructions that the logic circuitry in computers and other devices follow. Important types of software are application software for computer users (e.g. computer games and word processors), system software at the interface of hardware that support application software (e.g. operating systems), and middleware which forms a link between software components that have to function together, perhaps running on different operating systems. Software integrated with hardware that is instructed by it is called embedded software. This type of software is often used in electronic equipments other than computers, such as telephones or car components.

[6] This section is largely based on Gallini and Scotchmer (2002) and Guibault and Van Daalen (2005). See also Stiglitz (forthcoming).

[7] A more general question is whether there exist mechanisms next to IPRs that aim at maximizing the surplus generated by innovation. Examples of such mechanisms are prizes and procurement.

[8] Even though we focus on copyrights, *mutatis mutandis* the thrust of our economic argument related to innovation incentives and market also applies to patents.

[9] The *Auteurswet* does not allow reverse engineering, except under specific circumstances (art. 45m). For more on reverse engineering, see e.g. De Cock Buning (1993).

[10] Trade secrets consist of not releasing the underlying source code except under very strict conditions in combinations with severe penalties in case of breach of agreement or confidentiality. They are not recognized (e.g. in international treaties) as IPRs to the same extent as patents or copyrights.

One can distinguish open source software (OSS) and proprietary software (PS). OSS is "software which has freely available source code enabling the licensee to inspect, use, improve, expand and distribute the source code" (Ministry of Economic Affairs, 2007, p. 6).[11] This description requires some background. Software programs can be distributed as source code – which is what users need to have if they want to modify a program – or as object code. A program in source code must be compiled into object code before it can be executed. In the case of OSS, the underlying knowledge of a program as embodied in source code is publicly available for imitation, improvements and modifications, under certain conditions as specified in the accompanying license. Usually this is not the case for PS, for instance, because the source code is withheld or because of exclusive legal rights that preclude imitation and tampering. Note, however, that in principle nothing precludes a developer issuing software through a proprietary license from disclosing the source code.

The link between the IPR regime and the licensing structure of software can be made more precise. A copyright regime grants developers an exclusive right to reproduce and distribute the underlying code, as well as to develop derived works.[12] OSS licenses grant, under conditions, certain permissions. OSS licenses use the copyright regime to ensure that the source code of the software is available free of charge, and that the user of the software has certain rights to distribute the original program and to make modifications and improvements. Note, however, that independently of whether the software is licensed under proprietary or open source regime, not complying with the license constitutes a violation of copyright law.

Recent insights from the economic literature

Recapitulating, from an economic point of view, the traditional idea behind IPRs is to protect the incentives for innovation by ruling out copying and imitation. Therefore, one could be led to think that OSS causes innovation processes to stall. As we will argue, this may be true in some cases, however, the picture may also look completely different. Let us here point at the fact that economists, Nobel Prize winners Joseph Stiglitz and Eric Maskin included, are adding new insights to the traditional notions. As Stiglitz (forthcoming) writes in a survey paper on the economic basis of intellectual property rights: "The [...] intent [of the intellectual property regime] is to provide incentives to innovate by allowing innovators to restrict the use of the knowledge they produce by allowing the imposition of charges on the use of that knowledge, thereby obtaining a return on their investment" (p. 104). Nevertheless, one can argue that "[...] many of the most important intellectual advances are not covered at all by the patent system" (p. 105). Bessen and Maskin (2004) make a similar observation related to the growth of the Internet: "[...] the economic model underlying this traditional argument is surprisingly limited. [Much] creative activity is not the work of lone creators. Rather, it is interactive and involves

[11] For a more formal definition, see the one proposed by Bruce Perens at http://www.opensource.org/docs/osd. See also Groenenboom (2002).
[12] McGowan (2005).

the contributions of many different parties. Indeed, innovation is often sequential, where each creator improves on the work of the previous iteration" (p. 2). The basic idea of this observation is that in the standard model, copying and imitation are equally "bad". However, when the innovation process is interactive and sequential, imitation is not the same as copying, but it ultimately helps to create value for society.

3 Characteristics of open source and proprietary software

In order to understand whether there is a role for the government to intervene, in this section we describe the most important characteristics of software markets that may influence the level of competition and innovation. We compare proprietary and open source business models in order to understand (1) what drives developers of software, (2) what kind of market failures might be present, and (3) how these two characteristics affect welfare in the absence of government intervention.

3.1 Incentives to participate in open source software development

In this section, we seek to find answer to our first question: what are the incentives to innovate under different licensing regimes of software? We first describe the incentives for individual developers to participate in developing OSS without being paid, and then we discuss why profit oriented firms' may want to develop OSS.[13]

Proprietary software is sometimes perceived as the domain of private firms trying to make profits, while open source software is sometimes seen as the work of individual programmers, or by teams of individuals voluntarily participating in open source communities without being paid to do so. Examples of private firms developing and marketing well-known proprietary programs are Microsoft, Oracle, Sun Micro Systems and Apple. A well-known developer of open source software is Linus Torvalds, the initiator of the Linux operating platform. This platform is an example of a software developer community in which individuals contribute to the development of a certain program (such as Apache and Linux).[14]

Private firms may sometimes disclose the source code through an open source license.[15] Indeed, roughly half of all open source workers were paid to contribute to open source projects (Ghosh et al., 2002), implying that firms directly or indirectly supported them. Also, firms sometimes actively initiate OSS development, such as Sun Microsystems with Open Office. Of course, also an individual software developer may be primarily driven by profit maximizing incentives (i.e., acting as an entrepreneur who aims to develop and sell software under proprietary copyright protection).

3.1.1 Why do individual software developers voluntarily contribute to OSS projects?

Why do individual software developers voluntarily, without directly being paid to do so, contribute to OSS projects? Apparently, incentives other than pecuniary ones play an important role. To understand the links between OSS and innovation incentives, it is important to pinpoint what drives individual developers not employed by firms.

[13] This section is based on Lerner and Tirole (2005a and 2005b), Maurer and Scotchmer (2006) and Mendys-Kamphorst (2002).

[14] See Von Hippel and Von Krogh (2003).

[15] See Lerner et al. (2006).

Throughout this document, we adopt the common economic point of view that individual developers are rational and aim to maximize their utility. This also applies to their decisions to develop software. A programmer, when participating in software development, faces a variety of benefits and costs.[16] He decides to participate in a project if his benefits outweigh his costs. Note that this assumption does not imply that only monetary or material incentives matter: utility can have both extrinsic and intrinsic components. When someone's actions are extrinsically motivated, he or she often requires an external incentive, usually a monetary pay-off, to perform these activities. A person who is intrinsically motivated, however, derives utility from the mere activity itself and does not need such an external incentive.

Software developers working on proprietary projects receive direct payments in the form of wages or completion- or performance-contingent rewards. However, programmers working on open source projects do not necessarily receive such direct payments. In addition, they typically protect their contributions under a license that makes it difficult to benefit from the resulting product via selling it. Therefore, OSS developers must be driven by other incentives than direct monetary payments to exert effort. These incentives may be both intrinsic and extrinsic in nature.

There are several empirical studies (Alexy and Leitner, 2008, Hann et al., 2004 and Mockus et al., 2002 about the Apache project, Ghosh et al., 2002, Lakhani and Wolf, 2005, Henkel, 2005 and Van Wendel, 2005) and theoretical papers (e.g. Lerner and Tirole, 2005a) which investigate what drives software developers to contribute to open source projects. These studies report that developers primarily participate in open source projects due to extrinsic incentives, but a significant part of participants mentioned some intrinsic elements as their relevant motivation. There is still little research on the relative importance of intrinsic incentives as compared to extrinsic ones.

- The key *extrinsic motivations* discussed in these studies are: (i) own use benefits, (ii) job market signalling ("peer recognition" and "professional status enhancement"), and (iii) self-education ("skill enhancement"). We should note here that (ii) and (iii) may also be motivations for programmers working on proprietary software, and in general relate to future monetary pay-offs. As Lerner and Tirole (2002, 2005a) claim, the stronger these long-term incentives, (a) the more visible the product is to the relevant audience (peer, potential employer and users), (b) the higher the impact of effort on performance is, (c) the more informative the performance about talent is, and (d) the larger the network effects of programmers are.
- *Intrinsic motivations* have a social psychological nature and do not relate to monetary payoffs. Altruism, fun, meritocracy and ideology (reciprocity, gift benefit) often play an important role when a programmer chooses to participate in open source development. One should note that

[16] The programmer's costs include an opportunity cost of time, i.e., the value of the best alternative. Think e.g. of the forgone monetary compensation from an alternative job or the value of leisure time.

paid software developers may also be intrinsically motivated, but, arguably, less so than unpaid OSS developers.

3.1.2 Why do firms develop OSS?

In the discussion that follows (and throughout this document), we assume that a private firm maximizes profits. In general, a software-developing firm may generate revenues from selling the software or related complementary products and services. It faces initial software development costs, which can be high or low. High development costs may for instance arise if the development, besides programming skills, requires specific inputs such as high quality human capital, that have to be purchased at a high price. After the initial development cost has been incurred, the incremental cost of producing and selling copies of a program is negligible.

There are many examples of profit-oriented firms developing OSS. A firm will of course only initiate or participate in an open source project if it expects to make a profit from doing so. Therefore, the (expected) financial benefits of developing source code under an open source licence should outweigh the costs of giving up the profits that arise from selling the software under a proprietary license. For this condition to be satisfied, there are two requirements that open source projects initiated by private firms should meet. First, the chances of successful open source implementation should be big enough. The success of an open source project depends on the projects' ability to attract contributors. Not all types of software are equally suitable for open source implementation. It does not always make sense to 'go open source'. Second, the firm should be able to make money out of it. Therefore, two important questions are:

1. What types of projects are suitable for open source implementation?
2. How does OSS create revenues?

What types of projects are suitable for open source implementation?

Private firms developing open source software face several challenges with regard to successful implementation.

1. A first requirement is that it must be possible to implement the project as an open source project. As explained in the box "Consequences of the nature of developers' incentives for the organization of OSS projects", common characteristics of successful open source projects are modularity and leadership. Note that within the setting of a firm, the leadership condition will be much less of an issue than in more "organic" environments. Another remark relates to the quality of software. It is difficult to make unambiguous statements about quality of OSS in comparison to PS, as there does not exist consensus on this issue. On the one hand, software may be of higher quality if it is developed under single control, as with PS. This type of control is sometimes lacking in the case of OSS when a developer takes responsibility for his own

subproject.[17] On the other hand, one may argue that a hierarchical organisation can never do better than a network of developers.[18]

Consequences of the nature of developers' incentives for the organization of OSS projects

The success of an open source project such as Linux depends on the projects' ability to successfully attract software developers and eventually market the product. The organizational structure of an open source project will therefore be driven by the incentives of the individuals to join and participate in the project. Two key features of the organization of open source projects that are often mentioned are modularity and leadership.

Modularity refers to the way a software development project is organized. It requires that the project can be subdivided into separate modules, on which programmers can work independently. Modular organisation has two effects. First, it allows individual contributors to signal their capabilities more effectively. Due to the modular organisation of a project it is easier for outsiders to distinguish who contributed what to a project, and how important this contribution was. Second, modular design of projects reduces potential coordination problems that arise in open source development. A lack of coordination between programmers may for instance lead to wasteful duplication of efforts. Defining modules is an effective way to coordinate different contributions to an OSS project, by making transparent who works on which module. However, as Giuri et al. (2008) argue, a higher degree of modularity must be accompanied with more and effective leaders that control the project.

Leadership is relevant in the beginning of an open source project and when the initial idea is about to be launched. Firstly, a leader has to convince software developers to join the project. It is sometimes mentioned that successful open source projects should offer enough challenging programming problems that attract software developers (Lerner and Tirole, 2002). Secondly, even when open source projects are modularized, the governance of the community may be just as complex as in the case of commercial firms. Finally, a leader is important in marketing the software, particularly because OSS developers may not be primarily motivated by average consumers' needs. Projects are therefore more likely to be successful if someone takes responsibility in collecting information, deciding which contribution is relevant, and how the end product has to be marketed.

Leadership can be realized in different ways. For example, Linux benefited from an undisputed leader who initiated the project and kept his authority strongly centralized. In other cases, such as Apache, a committee makes decision through a consensus mechanism (Lerner and Tirole, 2002). Debian, an operating system based on Linux code and one of the oldest open source projects, was originally directed by a strong 'visionary' leader and supported by a community. Then its organizational structure shifted to an informal control mechanism based on a constitution, a yearly elected project leader, developers (together the "core" that is responsible for exploration aiming at innovation and development), and the maintainers (the "periphery", responsible for exploitation of existing skills) (Mateos-Garcia and Steinmueller, 2008).

[17] For more on the underlying complexity, see e.g. Lerner and Tirole (2002). There are also counterexamples where high quality software has been developed under single control, as for instance is the case of Linux, Mozilla's Firefox or Sun Microsystems' Open Office.

[18] For instance, as Raymond (1999) argues, "given enough eyeballs, all bugs are shallow".

2. Second, firms must be able to commit to keep the source code disclosed, instead of reverting to a proprietary regime once the project has become a success and the open source software has been adopted by a large number of users. One solution is for the firm to surrender control over the open source project to outside leaders' (Maurer and Scotchmer, 2006). In this way, if the private firm abandons the project, the same or another open source community can in principle continue to develop the software. This commitment problem can also be avoided by the proper choice of an open source licence, for instance if it implies that once a project is open source, it cannot be taken out of the open source domain (as in the case of GPL). An example may be Netscape's initiation of the Mozilla project,[19] making part of its (initially proprietary) browser source code freely available under the Mozilla end-user license agreement. Netscape initiated this project when it became clear it had lost the battle with Microsoft's web browser. This project was problematic from the beginning, which, as Lerner and Tirole (2002) argue, may have been due to doubts about Netscape's commitment to the open source nature of the project (the original license allowed proprietary derived works) and the project's insufficient modular nature. Mozilla finally disclosed the source code under licenses that permit anyone to modify it and the project eventually led to the open source web browser Mozilla Firefox, which is currently the second most popular internet browser.
3. Third, certain types of software require complementary skills, which are not readily or not at all available amongst software developers. If such skills are costly to procure, an open source license may not be appropriate because of the upfront capital requirement. Examples of such software may be games (requiring advanced graphical artwork) and specialized programs such as enterprise resource planning programs (e.g. SAP, using specific accounting and business administration skills, and also user support by well-trained professionals). If software includes such a specific component, it is likely that there is a substantial cost involved for which financial compensation is unavoidable. In that case, a proprietary license is more effective to get the project started.

We note here that these types of challenges tend to be solved by players in the market, that have various organizational modes and licensing forms available that support open source development. Further, it is the third challenge, the upfront capital requirement when costly, specific inputs are needed for the development of a program that will play a major role when we discuss policy implications later on.

[19] Mozilla used to be the codename of Netscape. Now it is the brand name of the Mozilla Corporation that releases various applications under open source licenses.

How does OSS create revenues?

Firms developing software under a proprietary license derive market power from copyright law, because they have the exclusive right to market their products by selling their copyrights or licensing their products for use by others.[20] [21] In contrast, under open source licenses firms do not derive market power from their license because everybody can use, improve and distribute the product for free. Profit maximizing firms will therefore only invest in open source software if this is profitable for other reasons, or if they derive market power from other sources.

One possible business model for commercial firms is what Lerner and Tirole (2005a) call 'living symbiotically'. Suppose the open source product generates revenues in a certain complementary and proprietary activities (products or services). If the additional profits in these complementary activities outweigh the costs of 'going open source', then disclosing the source code can be an attractive strategy. Complementary activities may for example be technical support, education, training, but also increased sales of complementary and proprietary software.[22] Private firms must convince volunteer contributors that the project is not simply abandoned to the open source domain if complementary activities become a commercial failure. One way of convincing participants is by making sufficiently large investments in open source development.

Sometimes firms may switch from proprietary development to open source development. Such a switch may occur if the switching costs are low, but the potential gains are high. This is the case when a firm is lagging behind a proprietary leader but sees the possibility that its product as OSS will be successful, thereby expects increasing profitability in complementary segments (Lerner and Tirole, 2005a). Open source development may counter-weigh a dominant position of a competitor (effects of competition), or it may increase reputation and visibility revenues in the long run (Henkel, 2005). Because of the commitment signal, it also reassures potential users that the product will not be withdrawn. An example may be the previously mentioned Mozilla project. Another reason is that the initial development costs of OSS may also be lower, because of the non-monetary incentives of employees.

Another source of revenues may stem from reduced development costs, for instance because of external development support, including the maintenance of the software by OSS communities (knowledge spillovers, see in section 3.2), and debugging.

[20] For our purposes, we adopt the economic definition of market power, that is, market power is defined as the ability of firms to charge a price above marginal cost (or to charge a mark-up). Note the difference with legal interpretations of market power.

[21] The actual extent of market power depends on various characteristics of the market, such as network externalities, switching costs and economies of scale. See section 3.2.

[22] Increased sales of complementary and proprietary products may be caused by network effects (see section 3.2.1), but also by a greater willingness of consumers to adopt the proprietary software.

3.1.3 Summary

In this section, we answered our first question, what motivates software developers under different licensing regimes. The incentives to develop software tend to be different under proprietary and open source licensing. Proprietary licenses create the possibility for software developers to gain market power immediately. Their incentives to develop software stem to a large extent from the expected profits from selling their software. In contrast, under open source licenses, developers forgo the possibilities provided by the IPR regime that grant market power. As a result, expected profits do not come so much (or not at all, in the case of free software) from sales revenues, but from elsewhere. Firms may generate alternative financial revenues through complementary products and services, such as support activities, premium versions, consulting services or attracting and retaining valuable employees. Moreover, developers contributing voluntarily to open source projects may be driven by intrinsic, non-monetary motivation such as altruism, fun, meritocracy and ideology, or by extrinsic, often long-term monetary incentives such as own use, self-education and job market signalling.

3.2 Market failures related to software

Our second question asks what kinds of market failures are present in software markets. If market failures exist and remain uncorrected, market processes will typically not result in welfare-optimal choices of licensing type, prices, quality or level of innovation. For policy makers it is therefore important to understand potential market failures in software markets and the ways in which they reduce short-term and long-term welfare. Software markets typically exhibit the following market failures: market power due to customer lock-in and market power due to economies of scale and knowledge spillovers.[23] It has to be noted here, that once a product has established itself, a firm owning a proprietary licence may have market power "granted" by the IPR regime.[24] [25]

3.2.1 Market power due to customer lock-in

Firstly, consider market power that is created by customer lock-in. In software markets, customer lock-in typically arises from the following sources: network externalities and switching costs.

[23] Market failures described in this section is based on Farrell and Klemperer (2007), Schmidt and Schnitzer (2003) and Geest et al. (2000).

[24] See for instance Maurer and Scotchmer (2006).

[25] One could add another source of market power, namely the presence of 'applications barriers to entry'. This type of entry barrier occurs, e.g., if no one can compete with an existing operating system without having a sufficient number of applications running on it. To some extent, this situation may be seen as a case of indirect network effects, or alternatively, as a two-sided market (see e.g. Rochet and Tirole, 2006). One could take this a step further by noting that an operating system must be able to run virtually every piece of software on every piece of hardware. The analysis of this problem is beyond the scope of this study; see Larouche (2008) for more on the European Microsoft court case.

Network externalities

Customer lock-in can be caused by network externalities. Network effects arise when a consumer's benefit from using software increases with the number of other consumers that use the same or compatible software. In addition, a consumer's benefit can be proportional to the number of complementary applications. Network externalities exist if consumers, when choosing between technologies, do not take these network effects into account, that is, these network effects are not internalized.

Network effects can be present both on the demand side and the supply side of the software market.

- On the *demand side* of software markets, we can distinguish between two types of network effects. First, *direct network effects* arise when consumers value a particular piece of software more, the larger the number of other users. The benefits from using a particular piece of software increase with the number of other consumers that use the same or compatible software. Examples may be text editors such as Word. Second, consumers may value a particular software program more when a larger number of complementary applications exist. These effects are commonly called *indirect network effects*. Examples of indirect network effects are operating platforms, such as Windows and Linux: the more complementary software is available for a platform, the more attractive it becomes for consumers to use that platform. The presence of network effects on the demand side encourages existing consumers to continue to use the same product, or new consumers to choose a product with a bigger network (*customer lock-in*), thus creating entry barriers for new, potentially superior products.
- The *supply side* of software markets may also exhibit direct and indirect network effects. In OSS development, *direct network effects* may arise because the attractiveness of an open source project for software developers depends on the number of other contributing developers. Intuitively, more contributors imply a stronger signal for other potential programmers. A larger number of active contributors allow a developer to show that he can outperform a larger group of co-developers. On the other hand, a large number of developers may deter an individual from contributing, because his marginal contribution may be very small. *Indirect network effects* also appear on the production side of the software market. The potential value (to developers) of developing new software that is complementary to other software increases with the number of users. For instance, the more consumers use a particular platform, the more profitable it will be to develop software for that platform.

Inefficiencies caused by network externalities

Besides enhancing market power, network externalities may lead to several other inefficiencies (based on Farrell and Klemperer, 2007).

The most important one is *under-adoption* that results when network effects are not internalized: when consumers ignore others' benefits of their adoption decisions then, from a welfare perspective, the number of consumers adopting a certain technology may be too low. Over-adoption may also occur – see below on market tipping.

If indirect network effects are not internalized, it will lead to *under-provision of complementary software development*. Note that, to a certain extent, a monopolist with a proprietary license can internalize network effects relating to complementary products. Its exclusive rights enable the firm to increase its price above cost proportionate to the number of complementary products. In this way, the monopolist can appropriate users' benefits from an increase in the number of complementary applications or an increase in quality.

Network externalities can also *tip the market* towards a certain technology or standard, not necessarily the one offering the highest user benefits (net of network effects), creating a monopoly position for the seller of that technology. The resulting high monopoly price creates a deadweight loss, that is, a loss in welfare that occurs when demand is reduced due to a mark-up in the price. Market tipping can be due to coordination problems between adopters. Which network good consumers adopt depends on which good they expect to be adopted. If network good A is superior to network good B, but everybody expects good B to be adopted, the inferior product will be adopted. Note that once the market has tipped, demand is biased towards the leading product in terms of network effects irrespective of the relative product quality compared to competitors' offerings.

Network externalities may lead to *coordination failures*: consumers may coordinate on an inferior technology (recall video standards VHS versus Betamax), or lack of consumer coordination may cause splintering, that is, they adopt multiple competing and incompatible products in a market where one product would be more beneficial (Lerner and Tirole, 2005a).

Switching costs

Consumers may face substantial costs when they decide to switch to another technology, product or supplier. Switching costs can for instance occur because of the need to learn how to use a different product, the cost of buying new compatible software programs or the cost of searching for an alternative product.[26] [27] In some cases, switching costs may be immaterial, for instance because of the attachment to a certain technology. Zwiebel (1995) provides a rationale for this type of switching cost. He argues that managers, mainly for reputation and because of lack of information, may abstain from switching to an alternative solution that is superior to the industry standard or the product of a dominant firm ("no one ever got fired for buying IBM").

High switching costs cause *customer lock-in*, which results in market power for software producers with a first-mover advantage. High switching costs may therefore lead to higher prices and cause entry barriers for suppliers of new software products.

[26] For an extensive analysis on switching costs see Farrell and Klemperer (2007).

[27] Close standards in software markets restrict the exchange of files between users of different applications, therefore making switching to a new software product costly. The problem of closed standards is more pronounced when network effects are present.

3.2.2 Market power due to economies of scale

Software markets are often characterized by *economies of scale* because the development of the initial 'first copy' of a product entails a fixed cost, while the costs of producing additional copies are relatively low or even close to zero.[28] Interestingly, in software markets, the implication is not necessarily that production by a single firm is the most efficient outcome. Even if the cost of producing a first copy is large, it may be spread out over participants in a programmers' community. In addition, the initial development cost may be negligible when programmers are willing to exert effort without direct payments, as may be the case with open source software.

There is an exception which may justify restrictions on the use, modification and distribution of software included in the software license. The development of certain software types requires purchasing specific, complementary inputs at a substantial cost, such as expertise that understand complicated econometric methods (as in the case of advanced statistical software) or graphic designer skills (as in the case of games). When such specific inputs are necessary, proprietary licenses work better to provide possibilities financial recoupment of development costs.

3.2.3 Knowledge spillovers

Finally, market failures may exist when *knowledge spillovers* are present. Knowledge spillovers from innovations occur if R&D by one firm facilitates innovation by other firms, who achieve success by 'standing on the shoulders of giants'. Such spillovers may facilitate two types of innovation:

- 'Imitative spillover': a somewhat differentiated product based, largely through imitation, on the original technology; and
- 'Creative spillover': a follow-on innovation that extends the original technology to a significant extent.

First, other firms may imitate an innovator's product without compensating for the latter's R&D costs. As a result of such imitative spillovers, the innovator's profit level is reduced ex post. Thus, ex ante, its incentives to innovate in the first place will also be reduced if imitation is easy. Second, the innovation process in software markets is often complementary and sequential, involving efforts of many developers who improve on each others' contributions. Therefore knowledge developed by one firm may help other firms to develop new innovations that do not directly compete with the initial innovation, for instance because they create a new market niche. This type of creative spillovers increases the social value of an innovation. If a firm does not take into account the additional benefits due to creative spillovers when making investment decisions, there may be too little investment from a social viewpoint. This effect is

[28] Shy (2001).

particularly pronounced when upfront development costs are large: not taking into account creative spillovers may then have the effect that a socially desirable investment is not carried out because it is privately unprofitable.

Proprietary licensing addresses both types of spillovers: (i) it provides protection against imitation of an innovator's idea, and (ii) the innovator is able to appropriate surplus from third-party follow-on innovations through licensing. Accordingly, through the prospects of financial recoupment, proprietary licensing safeguards the incentives to exert development efforts. However, it is important to realize that the argument on follow-on innovations assumes that an innovator knows how to price its idea to other producers. As Bessen and Maskin (2006) argue, this may be problematic because of the information asymmetry between an initial innovator and potential follow-on innovators with regard to the latter's future profits. In particular, competitors who want to build on an existing innovation may have useful ideas about how such innovations can be achieved, that are not available to the original firm. In that case, strict intellectual property rights may slow down the speed of innovation. Thus, when this type of asymmetric information inhibits contracting for the optimal license fee, the socially optimal level of innovation will not be obtained.

The welfare loss that occurs when an innovator is unable to appropriate surplus from follow-on innovations need not occur under open source licensing. When a software license allows for the use of the source code by others, follow-on innovations can take place without hindrance. Thus, a switch from PS to OSS development may increase the level of innovation by facilitating knowledge spillovers, provided that the innovation process takes place in an interactive and sequential manner (see also Bessen and Maskin, 2004).

3.2.4 Summary

This section described the relevant market failures in software markets. Market failures, if they remain uncorrected, may lead to market outcomes that deviate from the socially optimal choices. Policy makers therefore have to understand and take into consideration what types of market failures are present and how they affect welfare. Software markets exhibit several important market failures: customer lock-in caused by network externalities or switching costs; economies of scale caused by the need to purchase specific, complementary inputs at a substantial cost; and knowledge spillovers.

Firstly, customer lock-in can have two reasons. The presence of network effects encourages existing consumers to continue to use the same product, or new consumers to choose a product with a bigger network, thus creating entry barriers for new, potentially superior products. Consumers may also face substantial costs when they decide to switch to another technology, product or supplier. High switching costs also lead to customer lock-in.

Secondly, the development of certain software types requires purchasing specific, complementary inputs at a substantial cost, asking for sufficient financial recoupment

possibilities of development costs. In such situations, proprietary licenses are more suitable to ensure these incentives and hence work better to protect innovation incentives.

Finally, knowledge spillovers from innovations occur if R&D by one firm facilitates innovation by other firms. If knowledge spillovers are not taken into consideration by innovators, the levels of innovation might be too low from welfare perspective. We can distinguish two types of knowledge spillovers based on the original technology: 'imitative spillovers' that create somewhat differentiated products and 'creative spillovers' that create follow-on innovations. Proprietary licenses provide protection against imitative spillovers and hence guarantee the financial recoupment of expensive development efforts. Proprietary licenses may also create possibilities to appropriate surplus from creative spillovers. However, due to asymmetric information between the innovators, socially optimal contracts between initial and follow-on innovators may not be feasible. Open source licensing bypasses this problem, allowing for creative spillovers to take place and increase the level of follow-on innovation.

3.3 Effects of incentives and market failures on welfare

This section explores how market failures and incentives to develop software affect welfare. With respect to market power, there is a potential trade-off in software markets.

3.3.1 The classical trade-off

The classical trade-off related to the economic foundations of intellectual property rights is as follows:

- On the one hand, the prospect of market power may be necessary to provide the incentive to innovate in the first place. In particular, when software development entails large costs, innovation will only take place if these development costs can be recouped. Market power creates the necessary rents.
- On the other hand, once an innovation has been realized, prices should be at the level of marginal costs and entry barriers as low as possible, while knowledge spillovers should be as large as possible. Once software has been developed, users benefit from marginal cost pricing (that is, close to zero), seamless switching to other software and other developers benefit if the software can be freely modified to realize follow-on innovations.

This trade-off between market power on the one hand and low prices and entry barriers or more follow-on innovation on the other, is the well-known trade-off between ex ante incentives and ex post efficiency (welfare).

To a certain extent, the discussion on whether open source or proprietary licenses are optimal boils down to the trade-off described above. But there is more to it, as the work by

Bessen and Maskin (2004, 2006) on interactive and sequential innovation processes shows. In general, there is no clear-cut optimum. At the one extreme, one may claim that a certain level of market power is also present under open source licenses, and that this is enough to compensate for development costs. Others[29] argue that intrinsic motivations, own use, self-education, reputation, and job market signalling provide sufficient incentives. In these lines of reasoning, software that is developed under proprietary licenses would also be developed under open source licenses. In that case, proprietary licenses bring no benefit compared to open source, and open source would clearly dominate from a welfare perspective. At the other extreme, one may argue that under proprietary licenses the innovator can charge the optimal license fee for follow-on innovators, and therefore profitable follow-on innovations will be realized. In that case, open source licensing across the board would not necessarily bring any benefits compared to proprietary licenses, but would rather entail social costs if certain types of software would no longer be developed. In this case proprietary licenses would clearly dominate.

Based on current insights (which are rather limited), it is wise to presume that there is some truth in both viewpoints, and that depending on the situation at hand and the characteristics of the specific software market, one may be closer to either of these points of view. More precisely:

- In the absence of proprietary licenses, some types of software may not be developed because the costs are too high and cannot be recouped. This is typically the case if the development of a specific type of software requires large funds upfront. Think for example of complementary, specific inputs – requiring skills other than software programming skills – that are painstakingly developed by specialists outside of the network of software developers that can easily be copied. For such types of software, there may be a substantial cost involved for which financial compensation (rather than "immaterial compensation") is unavoidable, so that it is best brought to the market as PS. Development of PS will be driven by expected profits, which is affected by consumers' willingness to pay, and thus by the benefits that consumers are expected to derive from it.[30] [31] In particular, when a firm aims at making revenues by gaining a big market share, it will have to offer software that appeals to a broad group of "unsophisticated" users (who are not programmers by profession), and thus take into account the needs of an average user. In the absence of proprietary licenses, innovation, particularly in the initial stage where consumers' needs do not yet play an important role, would be primarily driven by intrinsic motivation (altruism, fun, ideology) and extrinsic motivation (own use, education, signalling developing

[29] See the references in the section on incentives.

[30] Note, however, as Von Hippel (2002) argues, it is sometimes costly for firms to follow heterogeneous consumer needs, for example because it is hard to know individual consumers' preferences or it is impossible to differentiate prices.

[31] Of course, the incentives provided by PS may co-exist with some of the advantages of OSS, especially when there are informal or enforced standards to ensure interoperability and compatibility between applications and file formats.

skills) of individual software developers.[32] These types of incentives may not be sufficient to drive development when there is a need to purchase expensive, specific inputs from specialists.

- In the presence of a proprietary license, the market power derived from IPRs may be strengthened by network externalities, switching costs or economies of scale. Market power can unduly reduce welfare: (i) market power may lead to a significant deadweight loss resulting from reduced demand due to high prices; (ii) market power may facilitate anti-competitive behaviour, especially when switching costs and network externalities create entry barriers for newcomers in the market; and (iii) if licenses are too restrictive, certain socially valuable follow-on innovations do not take place under proprietary licenses (while they would occur under open source licensing, see Bessen and Maskin, 2006). Therefore in software markets in which there is no need to purchase expensive, specific inputs from specialists outside the software community, open source licenses would result in higher social welfare. Note, however, that because there may be less attention to customer needs under open source development, open source licensing might result in inadequate documentation,[33] insufficient user friendliness or poor integration between software products.[34] This may for instance happen in the early stage of development, when programmers are not (yet) interested in the commercial potential of their software. At the same time, it should be noted that there are many examples of more mature OSS offerings that perform very well with regard to customers' wishes.

3.3.2 Summary

In this section, we analyzed how market characteristics, such as market failures and incentives affect welfare. In a market with market power due to customer lock-in new entrants may have too little chance to enter, even if they offer a superior product at a lower price. However, from a welfare point of view, market power might be needed and proprietary business models tend to dominate open source business models if specific inputs (such as non-programming skills) are needed so that development costs are high, or there are no obstacles (such as information asymmetries) with regard to contracting on follow-on innovations, or both. However, if software development requires mainly programming skills and contracting about derived works is inefficient while a continuing stream of follow-on innovations is socially valuable, then open source business models perform better than proprietary business models.

[32] See Maurer and Scotchmer (2006).

[33] Inadequate documentation might be a negative consequence of OSS development, for instance due to its limited signalling value. A common way to substitute the lack of documentation is to use search engines and other discussion forums on the internet that make it easier to bring together different experiences and practices.

[34] An example of poor integration between software products is the e-mailing and agenda software of Mozilla that are only partially integrated, unlike the integrated functions of Microsoft Outlook. Nevertheless, the fact that OSS typically uses open standards facilitates integration with other products. See Lerner and Tirole (2005a) for more on the incentives to develop interfaces.

4 The potential role of the government

In general, market failures lead to inefficient market outcomes (for example prices that are too high, consumers cannot switch, quality that is too low, or a suboptimal level of innovation). In section 3.2 we assigned a central role for the following market failures: customer lock-in caused by network externalities or switching costs; economies of scale caused by the need to purchase specific, complementary inputs at a substantial cost; and knowledge spillovers in relationship to the scope for contracting on follow-on innovations.

To correct for these market failures, policy intervention may be desirable. The question in this chapter is therefore what policy makers can do to alleviate market failures and enhance welfare in software markets. We assume that the government acts in the public interest, that is, it aims at maximizing welfare. We define welfare as both short-term welfare (static efficiency) and long-term welfare (dynamic efficiency). We also discuss different types of government failure that may arise either because the government makes mistakes, is not able to commit, or because additional objectives play a role (resulting in regulatory capture and rent seeking).

To address market failures and enhance welfare, several policy options are available: stimulating demand, stimulating supply and lowering entry barriers. This can be done in various ways, as will be seen later in this section. To assess what role the government could play in stimulating software development under an open source licensing regime, we address several questions:

1. Depending on the conclusion drawn in chapter 3, (i) how does stimulating OSS affect short- and long-term welfare, (ii) when is stimulating OSS recommended, and (iii) what are the policy options if intervention is needed?
2. What negative side-effects might arise as a result of policy intervention (i.e., what are potential forms of government failure)?

4.1 Policy implications

Whether intervention is desirable depends on which licensing regime dominates from a welfare perspective, if any, and which market outcome can be expected to arise in the absence of government intervention.[35]

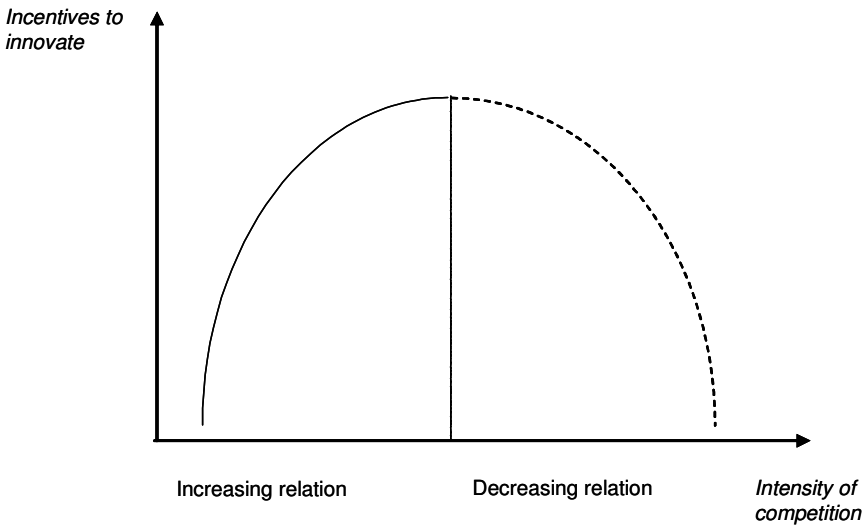
As we argued earlier, the case for government intervention in the software market is a subtle one. To derive policy recommendations we consider whether stimulating OSS enhances or reduces short-term welfare and long-term welfare. These welfare concepts are defined as follows.[36]

[35] Based on Schmidt and Schnitzer (2003) and Lerner and Tirole (2005a).
[36] See Bennett et al. (2001) for definitions of static and dynamic efficiency.

Short-term welfare or *static efficiency* can be measured by the surplus level when technologies are taken as given, that is, by how efficiently inputs are allocated to a particular product (productive efficiency) and the extent to which consumer needs are satisfied given that producers make sufficient profits to remain in the market (allocative efficiency). Higher static efficiency may for instance show up in lower prices due to more intense competition and lower entry barriers. Long-term welfare or *dynamic efficiency* is the aggregate surplus level over time when technologies may change, for instance due to innovations. Higher dynamic efficiency in the software markets may materialize mainly through product innovation (new products), increased choice for consumers and better quality products.

There may be a trade-off between competition in the short term (i.e., static efficiency) versus innovation incentives in the long term (i.e., dynamic efficiency). This trade-off can be illustrated by an inversed U-shaped curve (figure 4.1).[37] Because of this trade-off, in principle it is possible that too much competition may reduce dynamic efficiency. This may occur, for instance, if the market is already competitive to a large extent, so that further increasing the intensity of competition would lead to cut-throat competition, eroding the margins needed to invest in innovation (see right-hand side of figure 4.1). However, if there is a serious problem of customer lock-in, there is typically a low intensity of competition to start with (i.e., the market is situated at the left-hand side of figure 4.1). In such a situation, increasing competition tends to go hand in hand with increasing the incentives for innovation, and will therefore increase dynamic efficiency.

Figure 4.1 "Inversed U-shaped" relationship between intensity of competition and innovation incentives



[37] See Aghion and Griffith (2005).

When there is a trade-off between static and dynamic efficiency, we assume that outcomes that increase dynamic efficiency at the expense of static efficiency generally outperform the outcomes that increase static efficiency at the expense of dynamic efficiency. This is a natural assumption to make, given that – when the future is not too heavily discounted – the long run should get more weight than the short run. Also, increased innovation will eventually stimulate static efficiency.[38]

4.1.1 How does stimulating OSS affect welfare in the short and long run?

Below, we first discuss the effect of stimulating OSS on static efficiency, which is relatively straightforward. We then discuss the effects on dynamic efficiency, which is more complex. For the sake of argument, the discussions below are based on the assumption that the starting point is a market situation in which PS dominates the scene, while OSS plays a minor role. This is a natural point of departure to assess whether it makes sense to stimulate OSS.

Static efficiency

In the short run, stimulating OSS (either at the demand or supply side) typically increases the intensity of competition, since it introduces downward price pressure on proprietary products.[39] [40] Software users will not be willing to pay high prices if a comparable product can be obtained for free. By (+) we denote cases when stimulating OSS only affects prices (in a downward direction). In addition, when there are network externalities or consumers face high switching costs, stimulating OSS, for instance by acting as a lead customer (see section 4.1.3 for more details), providing sufficient incentives for developers to increase quality and stimulate compatibility, may help to overcome entry barriers caused by customer lock-in effects, thus further increasing static efficiency (denoted by ++). To conclude, stimulating OSS is always good for static efficiency, as it intensifies competition in the short run (see table 4.1).

Table 4.1 Effects of stimulating OSS on static efficiency: increasing level of competition

Presence of network externalities and/or high switching costs	Consequence of stimulating OSS on proxies (price, customer lock-in) of static efficiency	Overall effects of stimulating OSS on static efficiency
No	• Lower prices	+
Yes	• Lower prices • Reduction of entry barriers and alleviation of customer lock-in	++

[38] Nonetheless, the government may have different goals. For instance, it may prefer increasing short-term welfare in the first place.

[39] This depends on the extent to which OSS and PS offerings are substitutes. For instance, for a long time, Linux has not been seen as a fully fledged substitute for Windows.

[40] Price refers to overall usage costs, that is, including total cost of ownership.

Dynamic efficiency

Whether or not stimulating OSS increases or decreases dynamic efficiency is pivotal in justifying government policy. However, the effect of stimulating OSS on dynamic efficiency is ambiguous and more difficult to assess than the effects on static efficiency.

All else being equal, the fact that PS is sold at a (non-zero) price generates, through financial indicators, very direct feedback about customers' needs and desires for the seller. Thus, more information about demand characteristics is generated, especially compared to open source projects that are primarily driven by supply-side incentives such as intrinsic motivation and own use prospects.[41] In other words, PS projects tend to be better in developing products for a big mass of "average" or non-sophisticated customers (that is, consisting of users other than programmers) than open source projects that do not (yet) have the ambition to become a commercial success. Because of this, in a pure OSS world, certain types of programs would not be developed in the first place, in particular applications that are interesting for average users but not for programmers themselves. Interestingly, one can turn this argument around to construct a case in favour of OSS. Note that without commercial motives (again, think of supply-driven OSS projects), certain ideas get a chance to be developed that would not stand a chance to be adopted in a commercial environment (requiring prospects for financial revenues within a reasonable time horizon). At a later stage, it may turn out that a fraction of these ideas *do* lend themselves for commercial exploitation. Thus, OSS may help to "let a thousand flowers bloom", some of which may turn out to become mass market products at a later stage. Nevertheless, to avoid broad-brush sponsoring of a wide range of pet projects with all the inefficiencies that go with it, a potential need for government intervention arises only at a later stage, that is, when markets for some of these "flowers" have already come into existence.

Now one can make considerations related to dynamic efficiency more specific (see also chapter 3):

1. Certain types of software require a costly input of expertise complementary to programming skills (e.g. graphic design), that are painstakingly developed by specialists other than software developers and that can easily be copied. If development costs are high due to the need of such specific, complementary inputs, then a business model based on PS tends to lead to more innovation, since it is more suitable to get development off the ground. Stimulating OSS could then reduce dynamic efficiency.
2. If efficient contracting is possible, software developers with market power can provide (efficient) firms that wanted to develop follow-on innovation with the source code, in return for (a part of) the profits that these new products would generate. If this is the case, then stimulating OSS may not lead to more follow-on innovations than the efficient level. Nevertheless, efficient contracting of proprietary source code may not be feasible, for instance

[41] For commercial open source projects, e.g. in which profits are meant to be generated through complementary products and services, this argument does not apply.

because it is hard to observe whether contracting partners violate license conditions. Then stimulating OSS will lead to a higher level of follow-on innovation.

4.1.2 When is stimulating OSS recommended?

To recapitulate from section 3.2, for policy implications, the following market failures play a central role:

1. Customer lock-in caused by network externalities or switching costs;
2. Economies of scale in relationship to a need to purchase specific, complementary inputs at a substantial cost;
3. Knowledge spillovers in relationship to a sub-optimal scope for contracting on follow-on innovations.

Note that each of these market failures may or may not occur in a market for a specific program, depending on its characteristics. Assessing which market failures, and to what extent, are present, is a necessary step in order to design appropriate policy.

We suggest the following hierarchy in assessing the market failures above, starting with the most problematic type of market failure that may occur in software markets. Policy makers should first consider whether serious customer lock-in is present in the market (market failure 1).[42] If so, it has to be assessed what type of business model, if any, is most appropriate in terms of dynamic efficiency (market failures 2 and 3).

1. Suppose first that there is no serious customer lock-in problem (market failure 1 above). Should the government then intervene in order to stimulate OSS? At the demand side, customers do not face serious hurdles to switch to new entrants once they appear on the scene. At the supply side, new entrants (whether they offer PS or OSS) do not face substantial entry barriers impeding them from competing with an existing software supplier. When such entrants are more innovative, they will be able to gain market share quickly if there is demand for a new product variety. Market processes are therefore likely to result in an efficient constellation of licensing types. In principle, there is then no need to interfere with the market-driven emergence of innovations. Abstaining from ex ante intervention is then the best option, to avoid distorting market processes (see also under section 4.2 Government failures).
2. Second, suppose that there is a serious problem of customer lock-in. Then, in a market dominated by an incumbent offering PS, new entrants may have too little chance to enter, even if they offer a superior product at a lower price. Now the question is whether there are indications suggesting that, from a welfare perspective, PS may be a superior business model

[42] To asses how serious customer lock-in problem is, one can, for instance, consider whether switching costs are large. Econometric analysis or market research can be conducted to assess these costs. For that an analytical framework and case studies are presented in Pomp et al. (2005).

compared to OSS. More precisely, the question is whether it makes sense to support OSS in one way or another. The central point is that from a welfare viewpoint, OSS is not the best business model in all circumstances (this refers to market failures 2 and 3 above, also as is discussed in the previous section). If (i) software development requires a specific, complementary input at a substantial cost, while (ii) there are no obstacles (such as information asymmetries) with regard to contracting on follow-on innovations, then there are no strong reasons to assume that an open source business model performs better as a better business model to get development off the ground than PS. Now what do these conditions imply?

- If conditions (i) or (ii), or both, are satisfied to a substantial extent, it may *not* be wise to introduce policy specifically aiming at stimulating OSS. Actively stimulating OSS in such markets conflicts with the effectiveness of PS-based business models, and can lead to distortions of dynamic efficiency. However it should be noted here that *generic*[43] policy to reduce serious customer lock-in could be very beneficial, at the condition that more competition does not come at the cost of reduced innovation and dynamic efficiency.
- If neither condition (i) nor condition (ii) is satisfied, besides enhancing competition by generic instruments, directly stimulating OSS (by *specific* policy) may be very useful to help the market overcome customer lock-in.

It is also important to consider whether enhancing competition could go too far, in the sense that it would undermine innovation incentives (recall the discussion on the potential trade-off between static and dynamic efficiency, illustrated by figure 4.1). Theoretically this could happen if already at the beginning there is a high intensity of competition. However, when customer lock-in is present and incumbent suppliers experience little competitive pressure, it is likely that enhancing competition will stimulate innovation rather than reduce it.

Based on the arguments above, one can construct the following decision tree for policy makers.

[43] That is, reducing entry barriers for all types of software (not specifically OSS), such as mandatory use of open standards in government procurement.

Figure 4.2 Decision tree for policy makers

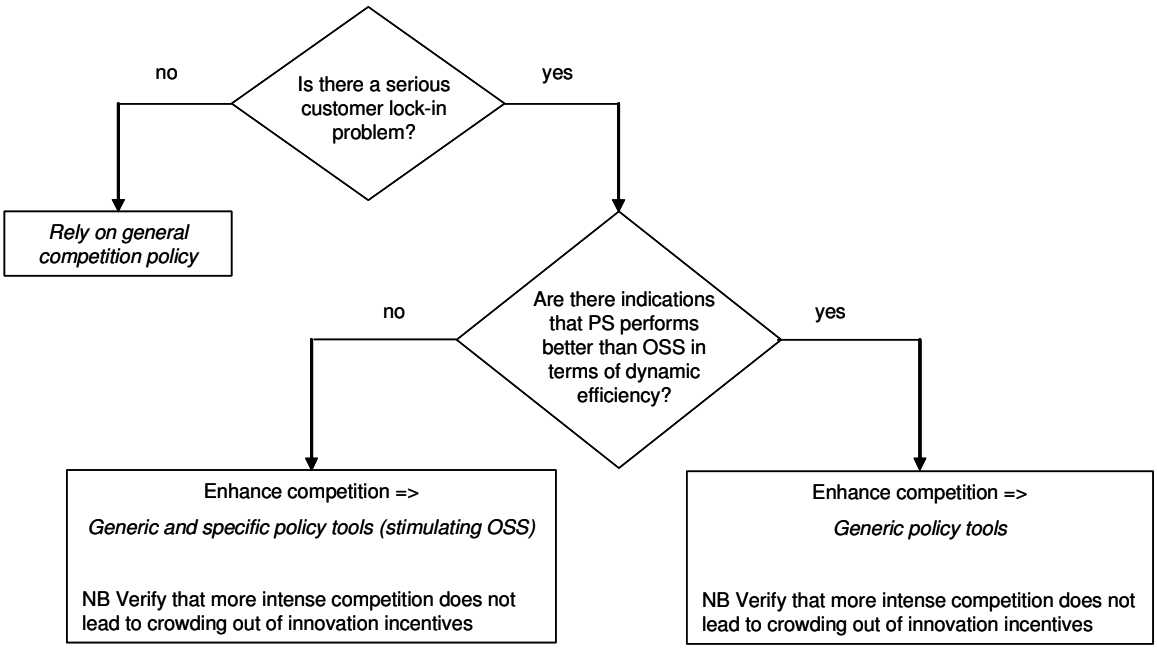


Table 4.2 illustrates the flow chart with some tentative examples. We stress that these examples are meant to provide a first intuition on how to use our framework, rather than a call for policy measures in these markets. Further analysis is needed in specific cases.

Table 4.2 Tentative examples of the decision tree

		Are there indications that PS performs better than OSS in terms of dynamic efficiency?	
		Yes	No
Is there a serious customer lock-in problem?	No	Advanced statistical software	Mathematical software, internet browsers, antivirus software, web content management software
	Yes	"High-end" interactive games, enterprise resource planning software	PC operating systems, standard office applications, enterprise content management software

Let us explain the examples in some more details. In the market of advanced statistical software (e.g. STATA, Eviews and SPSS), and mathematical software (e.g. Mathematica and its open source competitors Maxima or Sage), lock-in problems are less substantial than in markets where customers exchange files and applications on a continuous basis (e.g. operating systems and office applications). Hence network externalities form no substantial barriers to enter the market. Other entry barriers may of course exist, but those are likely to be less problematic.

In the case of advanced statistical software, development costs are high due to the need of specific, complementary inputs, that is, experts who understand complicated econometric

methods. Therefore PS can be more appropriate to provide for cost recoupment possibilities. Nevertheless, an entrant offering a superior alternative will be able to overcome the incumbent's position and recoup the necessary investment.

The development of mathematical software does not (or to a lesser extent) require such complementary, specific inputs. Therefore there are sufficient incentives to develop both OSS (Maxima) and PS (Mathematica), and due to relatively low entry barriers, competition between these software offerings can take place. Note, however, that OSS may lead to more follow-on innovation. Other examples are internet browsers (Internet Explorer and Firefox), antivirus software and web content management software (e.g. PS: Interwoven, FatWire and OSS: WordPress, Joomla!).

In the second row two groups of examples are given with serious customer lock-in problems. Firstly, of markets where PS performs better to increase the level of innovation. The market of high-end interactive games, in which players spend significant amounts of time on playing with each other, network effects are very strong, creating entry barriers for new games. Furthermore, the need to purchase advanced graphical skills makes the development costs high. Similarly, there may be high switching costs in the market of enterprise resource planning software (e.g. SAP and its open source competitor Compiere). Besides, software development requires specific inputs, like specific accounting and business administration skills, that increase development costs significantly. In these cases, PS-based business models may do a better job in cost recovery.

Secondly, PC operating systems, like Windows and Linux also exhibit substantial network externalities and switching costs. However there is relatively less need for specific, complementary expertise (beyond adequate programming skills) to develop such software. Thus there is no indication that PS performs better than OSS in stimulating innovation. By stimulating OSS in a way that it creates a significant installed base for rival offerings, consumers' barrier to switch can be reduced to a substantial extent. Other examples where this reasoning applies are office software (text editors, type setting applications, spreadsheets, and presentation editors) and enterprise content management software (e.g. Hummingbird and its open source competitor Alfresco).

As was pointed out, the discussion of these examples is tentative. A more complete analysis of particular cases is beyond the scope of this study.

4.1.3 Policy options if intervention is recommended

One can distinguish different policy options depending on whether they generally target software or specifically OSS. A policy option is *generic* if it aims at reducing entry barriers in software markets independently of type of software. When policy intervention directly aims at stimulating OSS, we classify this option as *specific*.

Let us first discuss generic policy tools. Firstly, mandating interoperability (in the sense of compatibility) between competing products can help to internalize network effects by allowing

consumers of different software programs to exchange files in a seamless manner.[44] This type of intervention may be particularly important when PS suppliers have incentives to hinder interoperability, for example by implementing closed standards in order to preserve their market power. The government can also facilitate interoperability if it acts as a coordinating medium. Secondly, to lower the perceived (non-financial) switching costs of consumers, the government could improve transparency of software offerings by facilitating the provision of information about the availability, price and location of products and relating services. Note that these types of ex ante policy intervention may also involve, to a certain extent, the application of competition law (ex post intervention), with the purpose of keeping the market open for innovation and ensuring that ultimately consumers decide which standard, technology or product will win.[45]

We now turn to specific policy tools. The government can stimulate OSS by acting as a lead customer, for instance by promoting or even requiring the use of OSS in the public and semi-public sector or by imposing OSS as a requirement in procurement. Note that large corporate clients may also more easily adopt OSS if purchasing managers can motivate their choice for OSS by pointing to the fact that well-established organizations also purchase OSS.[46] An example of specific policy is procurement for the development of a database management system that connects governmental agencies. Specific policy may be effective in "tipping" the market towards a viable market share for OSS. The underlying mechanism is that by doing so, the government realizes a significant installed base of users for an OSS alternative, making it more attractive for other consumers to follow.

The following table summarizes which policy tools are generic and specific. The table assumes that intervention is welfare enhancing – hence it applies only to cases where welfare effects are such that, overall, policy makers find it worthwhile to intervene in the first place. Note that the table does not make any statement about the relative effectiveness of the different tools.

Table 4.3 Generic versus specific policy interventions

	Generic policy tools	Specific policy tools
Ex ante intervention		
Lead customer with regard to OSS		✓
Requiring OSS in public procurement		✓
Enforcing compatibility between rival offerings	✓	
Increasing transparency of software offerings	✓	
Ex post intervention		
Competition law enforcement	✓	

[44] See also Lee (2006).

[45] For more on this in the light of the European Microsoft court case, see Larouche (2008).

[46] See Zwiebel (1995).

Finally, the following table exhibits the points where different interventions are applied, namely the level of the structure of the market, or the level of conduct of market participants.

Table 4.4 Points where government intervention is applied

Intervention affects:	Market structure	Market conduct
Demand of software	Requiring OSS in government procurement (note that this policy option also affects conduct, by way of the effect of the requirement)	Acting as lead customer with regard to OSS
Supply of software	Enforcing compatibility between rival offerings Application of competition policy	Increasing transparency of software offerings Application of competition policy

Note that this list is not exhaustive, but rather an overview of types of intervention that are as light-weight as possible. An exhaustive list may not exist, since the possibilities to intervene are virtually endless. In particular, our shortlist does not include any possibilities to subsidize demand (e.g. government sponsored distribution of OSS amongst citizens, teaching in schools) or supply (e.g. tax subsidies for software developers developing OSS, annual prizes for the best OSS product). A priori, such interventions are worthwhile to include when considering the range of potential options. Note, however, that besides risks of government failure associated with the distribution of subsidies (see below for more on government failure), subsidies introduce additional distortions. The reason is that they require that taxes are raised elsewhere in the economy to fund the subsidies, while taxation is typically distortionary in nature. This observation suggests that in order to compare policy options that may be applicable in a specific case, a cost-benefit analysis is desirable.[47] For the sake of conciseness, in what follows, we will stick to our shortlist of potential interventions, being aware that in those cases where a specific type of intervention is desirable, more options could be considered.

4.2 Government failures

It is important to acknowledge that "good intentions" are not sufficient to motivate government interventions, since they can have benefits as well as costs. Accordingly, to be able to conclude that a policy indeed corrects a market failure, one must ask whether it actually reduces the inefficiency or welfare loss that was caused by the market failure at hand.[48] In other words, one must take into account the possibility of government failures, which are failures that arise when the government introduces a new inefficiency "because it should not have intervened in the first place or when it could have solved a given problem [...] more efficiently, that is, by generating greater net benefits" (Winston, 2006, p. 2-3).

[47] It is outside the scope of this report to compare the pros and cons of the different options that are available.
[48] Winston (2006).

Unfortunately, unlike the standard list of potential market failures on which economists agree (market power, natural monopoly, imperfect information, externalities and public goods), there is no "received wisdom" with regard to potential government failures. Ultimately, the effects of intervention can only be assessed empirically. For the purposes of this report, however, it is useful to construct a shortlist of failures that should be taken into consideration ex ante, that is, before implementing a specific policy.

Government intervention that aims at directly stimulating OSS may introduce the following welfare distortions:

- When the government acts as a lead customer or steers the market in other ways, there is no guarantee that it makes a choice that is superior to the outcome of market processes or choices made at a decentralized level by market participants. This type of argument is a standard criticism in the economic literature on innovation policy, where economists usually recommend against policies that aim to pick or support "winners" (see e.g. Boone and van Damme, 2004). As Schmidt and Schnitzer (2003) argue in the context of software markets, independently of the presence of market failures, an inappropriate product choice could tip the market in the wrong direction. Thus, the cure may be worse than the disease. Also, by doing so, competition and innovation incentives may be reduced.
- Policy makers may not be able to asses properly the long-term success of a publicly supported OSS project, for instance due to incomplete information. As a result of public procurement biased towards OSS, a currently profitable provider of PS may lose its incentives to invest.
- Government procurement processes based on a bias towards a certain type of technology, which involves making choices between different projects or suppliers, invites lobbying by candidate suppliers, and typically introduces inefficiencies due to rent-seeking behaviour.

4.3 Summary

In this section, we addressed the question what policy makers can do to alleviate market failures and enhance welfare in software markets. In section 3.2 we assigned a central role for the following market failures: customer lock-in caused by network externalities or switching costs; economies of scale caused by the need to purchase specific, complementary inputs at a substantial cost; and knowledge spillovers in relationship to the scope for contracting follow-on innovations. We suggest a hierarchy in assessing these market failures, starting with the most problematic that may occur in software markets. Policy makers first have to consider whether serious customer lock-in is present in a specific software market. If so, policy intervention may be recommendable. However, it has to be assessed what type of business model, if any, is most appropriate in terms of dynamic efficiency. If software development requires a specific, complementary input at a substantial cost, while there are no obstacles with regard to

contracting on follow-on innovations, there are no strong reasons to assume that an open source business model performs better as a better business model to get development off the ground than PS. In this case generic policy options, aiming at lowering entry barriers, can be applied. Requiring interoperability (compatibility) and increasing transparency of software offerings are mentioned as such. If none of the above mentioned conditions hold, open source business models may perform better than PS in stimulating interactive and sequential innovation processes. Besides the previously mentioned generic policy tools, competition can then be enhanced by OSS specific policy tools, such as acting as a lead customer or requiring OSS in public procurement. Note that, in general, competition law enforcement may be necessary as a complementary type of intervention or oversight. Before implementing these policy measures, however, it is very important to realize that intervention may introduce government failure. Therefore, one should try to estimate the net welfare effects of intervention, taking into account that typically, the government is not an equally good arbiter with regard to technological choices as the market.

5 Conclusions and discussion

This study analyzes the economic effects of publicly stimulating open source software (OSS) on competition and innovation in software markets. To recall from the introduction, we raise several questions:

1. What are the incentives to innovate under different licensing regimes of software?
2. What are the market failures in software markets?
3. How do incentives and market failures affect welfare?
4. What can policy makers do to alleviate market failures and enhance welfare in software markets?

At the supply side, software markets may exhibit monetary as well as non-monetary incentives of software developers, and market power and knowledge spillovers as the potential market failures. When intervening in software markets, policy makers have to take into consideration that there exist trade-offs between encouraging follow-on innovations (ex post) versus preventing that others can free ride on existing innovations (which is good for ex ante incentives). The answer to the question whether proprietary or open source licenses are optimal is therefore not obvious or univocal. Based on economic arguments, we conclude that from a welfare point of view, proprietary business models tend to perform better if development costs are high due to the need of specific inputs and optimal contracting on license fee about derived works is feasible. In contrast, open source business models tend to dominate proprietary business models, if development requires mainly programming skills (which may be very advanced) and creative spillovers are socially valuable while proprietary licenses are too restrictive to provide sufficient incentives for follow-on innovations. Note that these are stylized arguments – specific cases require separate studies that go into deeper details.

Before discussing the policy recommendations, let us stress that specific interventions require a strong motivation, the more so because of the risk of government failure. In practice, competition policy and making sure that consumers can ‘vote with their feet’, are the best candidates to counterbalance market power and reduce entry barriers. The reason is that consumers are better informed about their own preferences with regard to new products than the government. Nevertheless, there may be a case for additional, ex ante intervention. However, it should be verified that this type of intervention is not counterproductive with regard to competition law enforcement. A more precise analysis of policy interventions discussed in this document that complement competition policy, is beyond the scope of this study.

The extent to which ex ante intervention is advisable depends on the presence of market failures. First of all it has to be assessed whether there are serious customer lock-in problems in the market. If so, policy intervention may be recommendable, depending on which business model is more appropriate to safeguard or stimulate innovation incentives. If proprietary

business models perform better, generic policy options, aiming at lowering entry barriers, can be applied to reduce customer lock-in. Requiring interoperability (compatibility) and increasing transparency of software offerings are mentioned as such. Stimulating OSS can be recommendable when open source licenses dominate proprietary licenses with regard to dynamic efficiency. Policy options that can help to alleviate customer lock-in problems and increase dynamic efficiency are acting as a lead customer or requiring OSS in public procurement. Note that in general, competition law enforcement may be necessary as a complementary type of intervention or oversight.

We conclude this section with a brief observation. In this document we discussed the effects of policy intervention on the level of competition and innovation in software markets without incorporating country-specific characteristics, mainly because software markets can be considered as global markets. Nonetheless, to reflect briefly on Dutch software markets, it is unlikely that the Netherlands can have a strong impact on global software markets. Policies aimed at stimulating OSS, that can nevertheless help to create a significant customer base for OSS products within the Netherlands, should be seen in this light.

References

Aghion, P. and R. Griffith, 2005, *Competition and Growth: Reconciling Theory and Evidence*, MIT Press.

Alexy, O. and M. Leitner, 2008, Rewards, and Their Effect on the Motivation of Open Source Software Developers, Mimeo, Technische Universität München.

Bennett, M., P. de Bijl and M. Canoy, 2001, Future Policy in Telecommunications, An Analytical Framework, CPB Document 5.

Bessen, J. and E. Maskin, 2004, Intellectual Property on the Internet: What's Wrong with Conventional Wisdom? Available at: http://www.researchoninnovation.org/iippap2.pdf.

Bessen, J. and E. Maskin, 2006, Sequential, Innovation, Patents, and Imitation, NajEcon Working Paper Reviews.

Boone, J. and E.E.C. van Damme, 2004, Marktstructuur en innovatie, in: B. Jacobs and J.J.M. Theeuwes (eds.), *Innovatie in Nederland: De Markt Draalt en de Overheid Faalt*, KVS Amsterdam, pp. 71-92.

De Cock Buning, M., 1993, Auteursrecht en 'reverse engineering': techniek en theorie, *Intellectuele eigendom & reclamerecht,* 9(5), pp. 129-137.

Gallini, N. and S. Scotchmer, 2002, Intellectual Property: When Is It the Best Incentive System?, in: A. Jaffe, J. Lerner and S. Stern (eds.), *Innovation Policy and the Economy*, Vol. 2, MIT Press, pp. 51-78.

Geest, S.A. van der, R.C.G. Haffner, P.T. van der Schans and M. Varkevisser, 2000, Marktwerking en nieuwe ICT-markten: de markt voor software, Onderzoekcentrum Financieel Economisch Beleid (OCFEB) and Erasmus Universiteit Rotterdam.

Ghosh, R.A., R. Glott, B. Krieger and G. Robles, 2002, The Netherlands Free/Libre and Open Source Software: Survey and Study FLOSS, Mimeo, International Institute of Infonomics, University of Maastricht, available at http://www.infonomics.nl/FLOSS/report/.

Giuri, P., F. Rullani and S. Torrisi, 2008, Explaining leadership in virtual teams: The case of open source software, *Information Economics and Policy*, 20(4), pp. 305-315.

Groenenboom, M.M., 2002, Software licenties: van closed source tot open source, *Computerrecht* 1, pp. 21-29.

Guibault, L. and O. van Daalen, 2005, Unravelling the Myth around Open Source Licenses: An Analysis from a Dutch and European Law Perspective, Mimeo, Institute for Information Law, University of Amsterdam.

Farrell, J. and P. Klemperer, 2007, Coordination and Lock-In: Competition with Switching Costs and Network Effects, In: Armstrong and Porter (Eds.), *Handbook of Industrial Organization*, Vol. 3, Elsevier.

Hann, I-H., J. Roberts, S. Slaughter and R. Fielding, 2004, An empirical analysis of economic returns to open source participation, Mimeo, Carnegie Mellon University.

Henkel, J., 2005, Selective revealing in open innovation processes: The case of embedded Linux, *Research Policy*, 35, pp. 953–969.

Knubben, B.S.J., 2001, *Open Source Software, De Bron Geopend: Een economische analyse van Open Source Software*, Ph.D. thesis Faculteit der Economische Wetenschappen en Econometrie, University of Amsterdam.

Lakhani, K.R. and R.G. Wolf, 2005, Why hackers do what they do: understanding motivation and effort in free/open source software projects, In: J. Feller, B. Fitzgerald, S. Hissam and K.R. Lakhani (eds.), *Perspectives on Free and Open Source Software*, MIT Press, Cambridge, MA.

Larouche, P., 2008, The European Microsoft Case at the Crossroads of Competition Policy and Innovation, TILEC Discussion Paper 2008-021, Tilburg University, available at SSRN: http://ssrn.com/abstract=1140165.

Lee, J.-A., 2006, New Perspectives on Public Goods Production: Policy Implications of Open Source Software, *Vanderbilt Journal of Entertainment and Technology Law,* 9(1), pp. 45-112.

Lerner, J., P.A. Pathak and J. Tirole, 2006, The Dynamics of Open-Source Contributors, *American Economic Review*, 96(2), pp. 114-118.

Lerner, J. and J. Tirole, 2002, Some Simple Economics of Open Source*, Journal of Industrial Economics*, 50(2), pp. 197-234.

Lerner, J. and J. Tirole, 2005a, The Economics of Technology Sharing: Open Source and Beyond, *Journal of Economic Perspectives*, 19(2), pp. 99-120.

Lerner, J. and J. Tirole, 2005b, The Scope of Open Source Licensing, *Journal of Law, Economics, and Organization*, 21(1), pp. 20-56.

Mateos-Garcia, J. and W.E. Steinmueller, 2008, The institutions of open source software: Examining the Debian community, *Information Economics and Policy*, 20(4), pp. 333-344.

Maurer, S.M. and S. Scotchmer, 2006, Open Source Software: The New Intellectual Property Paradigm, In: T. Hendershott (ed.), *Handbook of Economics and Information Systems*, Elsevier, pp. 285-319.

McGowan, D., 2005, Legal Aspects of Free and Open Source Software, In: J. Feller (ed.), *Perspectives on Free and Open Source Software*, MIT Press, pp. 439-440.

Mendys-Kamphorst, E., 2002, Open vs. closed: Some consequences of the open source movement for software markets, CPB Discussion Paper 13.

Ministry of Economic Affairs, 2007, The Netherlands in Open Connection: An action plan for the use of Open Standards and Open Source Software in the public and semi-public sector, Dutch Ministry of Economic Affairs, The Hague. Available at http://www.ez.nl/dsresource?objectid=154648&type=PDF.

Mockus, A., R.T. Fielding and J.D. Herbsleb, 2002, Two Case Studies of Open Source Software Development: Apache and Mozilla, *ACM Transactions on Software Engineering and Methodology*, 11(3), pp. 309-346.

Pomp, M., V. Shestalova and L. Rangel, 2005, Switch on the competition. Causes, consequences and policy implications of consumer switching costs, CPB Document 97.

Raymond, E., 1999, *The Cathedral and the Bazaar,* O'Reilly Media.

Rochet, J.-C. and J. Tirole, 2006, Two-Sided Markets: A Progress Report, *RAND Journal of Economics*, 35(3), pp. 645-667.

Shy, O., 2001, *The Economics of Network Industries*, Cambridge University Press.

Schmidt, K. and M. Schnitzer, 2003, Public Subsidies for Open Source? Some Economic Policy Issues of the Software Market, CEPR Discussion Paper 3793.

Stiglitz, J.E., forthcoming, Economic Foundations of Intellectual Property Rights, *Duke Law Journal* (paper delivered at 2007 Annual Frey Lecture, Duke University).

Van Wendel de Joode, R., 2005, *Understanding open source communities: an organizational perspective*, Ph.D. thesis, University of Technology, Delft.

Von Hippel, E, 2002, Open source software projects as user innovation networks, Mimeo, MIT Sloan School of Management.

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), pp. 209-223.

Winston, C., 2006, *Government Failure versus Market Failure. Microeconomics Policy Research and Government Performance*, AEI – Brookings Joint Center for Regulatory Studies, Washington, D.C.

Zwiebel, J., 1995, Corporate Conservatism and Relative Compensation, *The Journal of Political Economy*, 103(1), pp. 1-25.